

CHAPTER IV

SYSTEM IMPLEMENTATION

Introduction

In Websim, the administrator has the access to design and manage a simulation system, they are granted with upload, change, delete and other administrative privileges. Meanwhile, an ordinary user is only allowed to execute the simulation system and view the results (as shown in Figure 11). After entering Websim, the user may select a pre-installed simulation project. Once simulation is done, the results will be plotted. An independent plotting program, Gnuplot will translate the ASCII input text file generated by the simulation program into a GIF image file. The CGI scripts then return an HTML page including the generated image. The overview of this system implementation is shown in Figure 21.

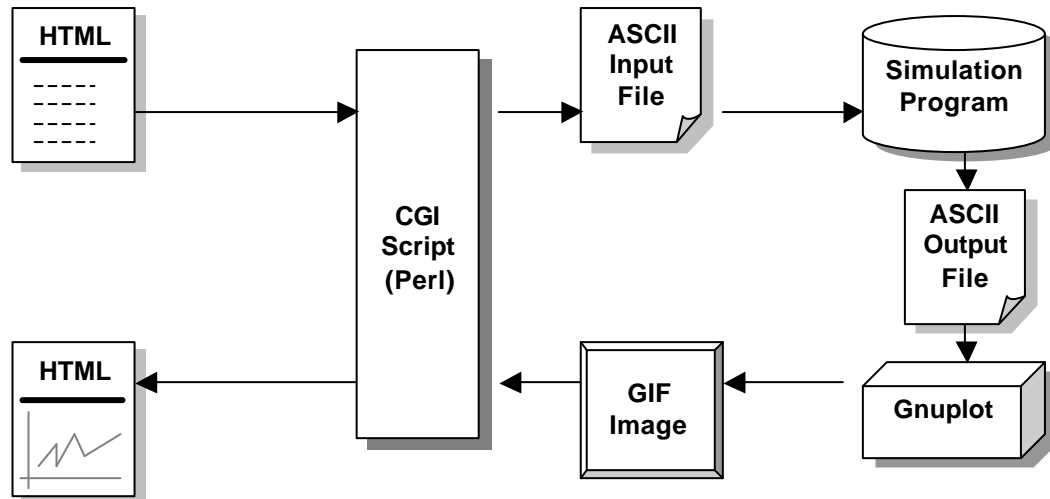


Figure 21: Block Diagram Of Websim System Implementation

This chapter describes the implementations of Websim. It is organized into three major sections. The first section discusses the implementation tools used in Websim development. The next section provides the detailed description of the applications aspects. Here three important Websim applications are described in details:

- (i) Generating simulation program web-interface.
- (ii) Performing simulation over the web; and
- (iii) Presentation of simulation results.

The last section of this chapter focuses on the security issues of Websim.

Implementation Tools

Websim utilizes CGI technology and uses Perl as the implementation language. The controlling CGI scripts, which coordinate the overall activities of Websim has been written in Perl. It is generally used for two purposes: data

collection and interactive communication. This is achieved by using Perl to generate HTML. Perl is by far the most widely used language for CGI programming and contains many powerful features, including:

- Highly portable and readily available.
- Contains extremely powerful string manipulation operators, as well as functions to deal with binary data.
- Makes calling shell commands very easy, and provides some useful equivalents of certain system functions.

Another language tool used in Websim is the Javascript. It is used because of its ability to interact with HTML forms and we would be able to perform verification of a form before it is submitted. Javascript can also perform preprocessing of input data, which can reduce the amount of data that must be transmitted to the server. In some cases, client-side Javascript can eliminate the need for CGI scripts on the server altogether. On the other hand, Javascript and CGI work well together.

Presentation of simulation results in graphical format is achieved by using a graph plotting utility named Gnuplot. Gnuplot is a command-line driven interactive function plotting utility for UNIX, MSDOS, and VMS platforms. This graphical program allows users to visualize mathematical functions and data. Gnuplot supports many different types of terminals, plotters, and printers (including many color devices, and pseudo-devices like LaTeX) and is able to generate a GIF file too.

Generating Simulation Program Web-Interface

We shall discuss the implementation issues related to generating web-interface HTML file for the simulation program in this section. First, the Websim administrator must have a Websim-compatible simulation program in his local hard disk. The simulation program must follow a certain format required by Websim (as mentioned in Simulation Program section, page 32), and compiled into a binary format (.exe). Websim does provide a template file for user to refer (as shown in page 33). Once this is done, simulation program has to be deployed on the Websim server. Here we present the steps in generating the web-interface file for the simulation program.

Step 1: Upload simulation program

Websim provides an upload simulation program service, which may be accessed from the main menu. When this service is called, Websim asks for an executable (.exe) file from the administrator and require him to specify the number of input and output parameters for the simulation program (Figure 22). The HTML tag to request for an upload file is shown below.

```
<INPUT TYPE="file" NAME="file_name" SIZE="30">
```

Web-based Simulation Environment

STEP 1: Upload EXE File

- To use this form, enter a title and choose a file to upload.
- File sizes are limited to 500K, larger files will return an **'Internal Server Error'**

File Type:	<input checked="" type="radio"/> - C/C++ Compiled Executable Files
Filename:	G:\k1\tc\temp\Watm.exe <input type="button" value="Browse..."/>
Parameters:	Input <input type="text" value="4"/> Output File <input type="text" value="2"/>

Figure 22: Uploading Simulation Program

Step 2: Upload status notification

Once the next button is clicked, the script tries to upload the local simulation program into the respective project folder. We first create a variable *\$simfile* to store the simulation program file's full path. The file is opened for output using the *MYFILE* file handle. The content of the uploaded file *\$file_name* is read and printed to the file handle. Once this process is completed, we close the file handle and the file is considered to be successfully uploaded to the server.

```
$simfile = join ("", $project{'dir'}, "\\sim.exe");
open (MYFILE, ">$simfile") || die $!;
while(read($file_name,$data,1024))
{
    print MYFILE $data;
}
close (MYFILE);
```

If the simulation program is successfully uploaded to the server, a confirmation message will be displayed on the screen, as shown in Figure 23.

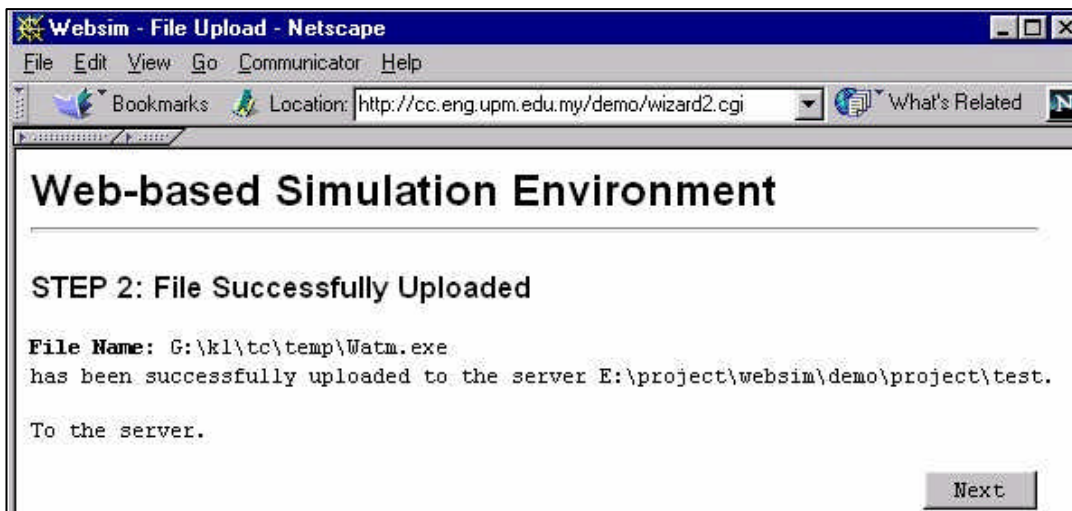


Figure 23: Confirmation on Successful Simulation Program Upload

Step 3: Enter web-interface settings

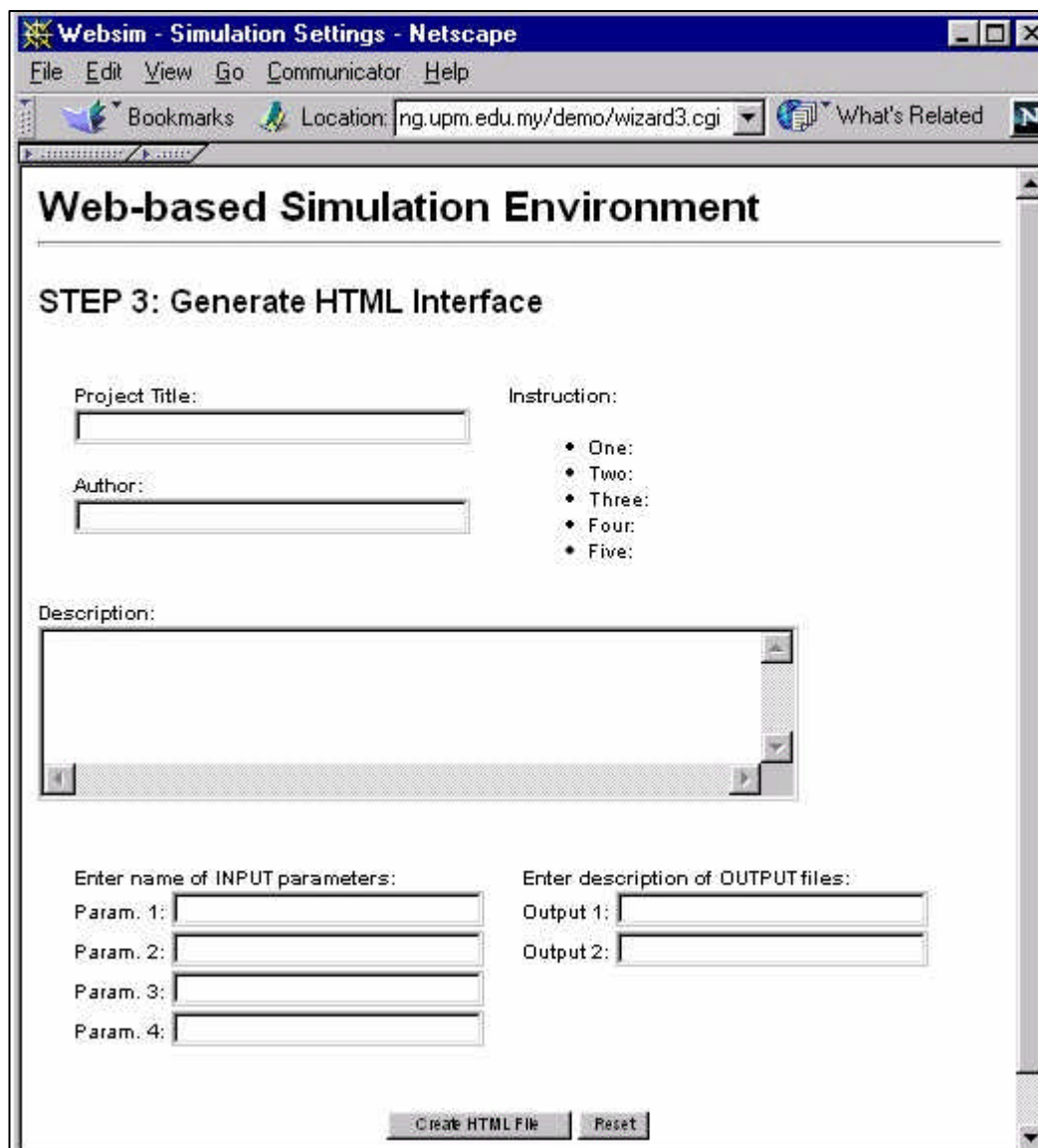
Here the administrator has to specify the web interface settings such as project titles, description of project, name of simulation inputs, etc. (as shown in Figure 24). The number of text fields for input parameters is determined in step 1, and the script to perform this task is shown below. The *\$in* variable store the number of input parameters and *input\$*n** variable is used to keep the name of the parameter.

```

#$in = number of input parameter
for ($n=1; $n<=$in; $n++) {
    $temp = $q->param('input'.$n);
    print <<HTML;
    Param. $n: <input type="text" name="input$n"
    value="$temp" size="25"><br>
HTML
}

```

All the data collected here will be used to create a HTML file, which will act as the web-interface for the simulation program.



The screenshot shows a Netscape browser window titled "Websim - Simulation Settings - Netscape". The address bar shows the URL "ng.upm.edu.my/demo/wizard3.cgi". The main content area is titled "Web-based Simulation Environment" and contains a form for "STEP 3: Generate HTML Interface".

The form includes the following fields and options:

- Project Title:** A text input field.
- Author:** A text input field.
- Description:** A large text area with scrollbars.
- Instruction:** A list of radio buttons with options: One, Two, Three, Four, and Five.
- Enter name of INPUT parameters:** Four text input fields labeled Param. 1, Param. 2, Param. 3, and Param. 4.
- Enter description of OUTPUT files:** Two text input fields labeled Output 1 and Output 2.

At the bottom of the form, there are two buttons: "Create HTML File" and "Reset".

Figure 24: Form Collecting Information for Web-Interface File

Step 4: Preview and finalize settings

Once all the setting form fields are entered, the user has to click on the "Create HTML File" button to create a new HTML interface file. Websim provides a preview function for users to preview the settings made earlier by just clicking on the "Preview" button. When this button is pressed, a new window appears showing the

final web-interface file. The final web-interface file contains not only the project name and descriptions, but also some output graph settings such as graph size, graph title, graph type, etc. At the bottom of the interface file is found a "Run Simulation" button used to trigger a script, which will execute the simulation program. Figure 25 shows a snapshot of the preview page.

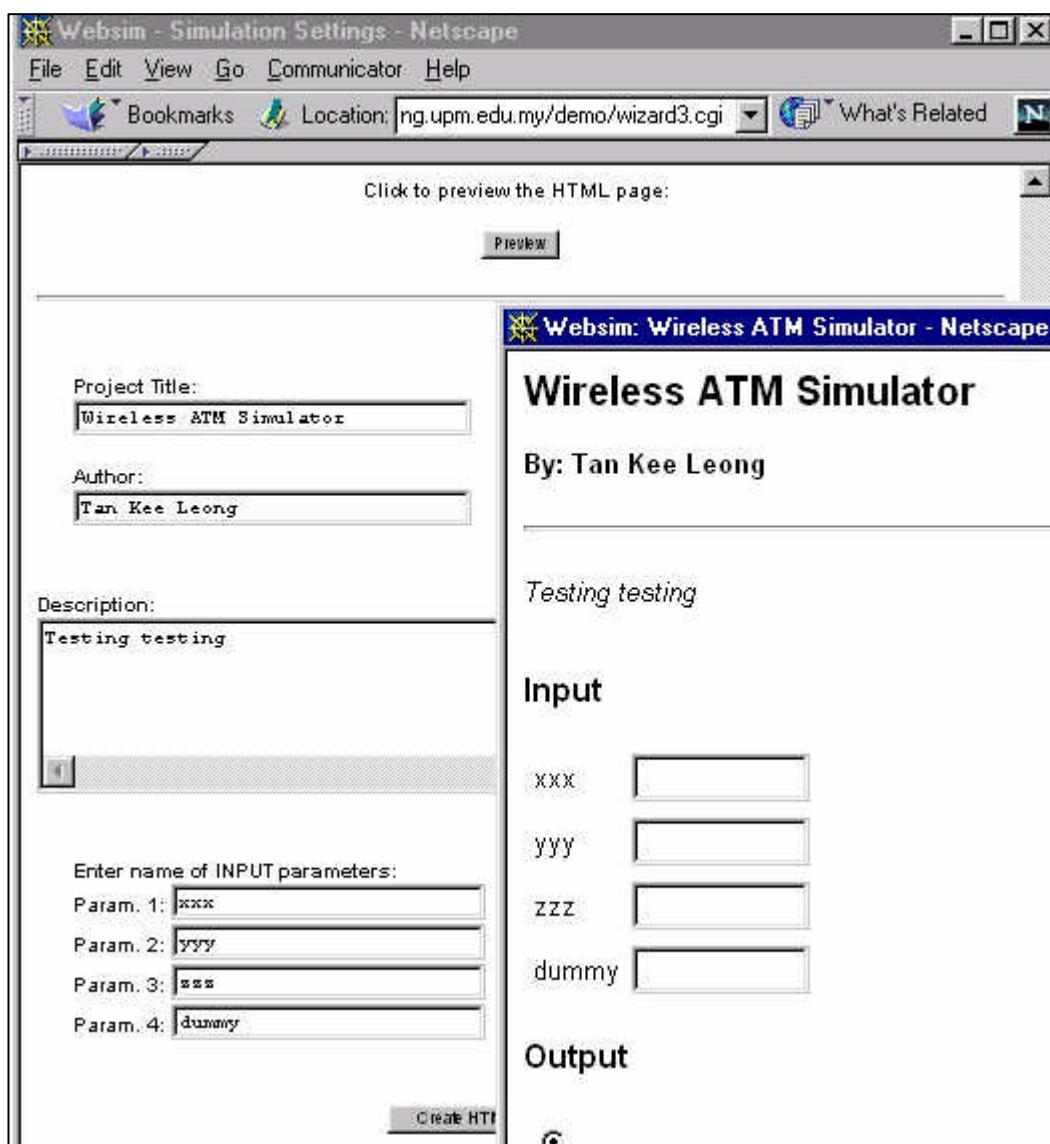


Figure 25: Preview of HTML Interface File

If a user chooses to further alter or edit certain settings, he may do so by entering the new settings on respective fields and click on the "Create HTML File" button again. Once satisfied, the user has to click on the "Finish" button to permanently save the file in the simulation project folder. The file is saved as a HTML file with the name "index.htm" so that it becomes the default page of the simulation project folder. Lastly, the URL of the simulation project folder is displayed (Figure 26).

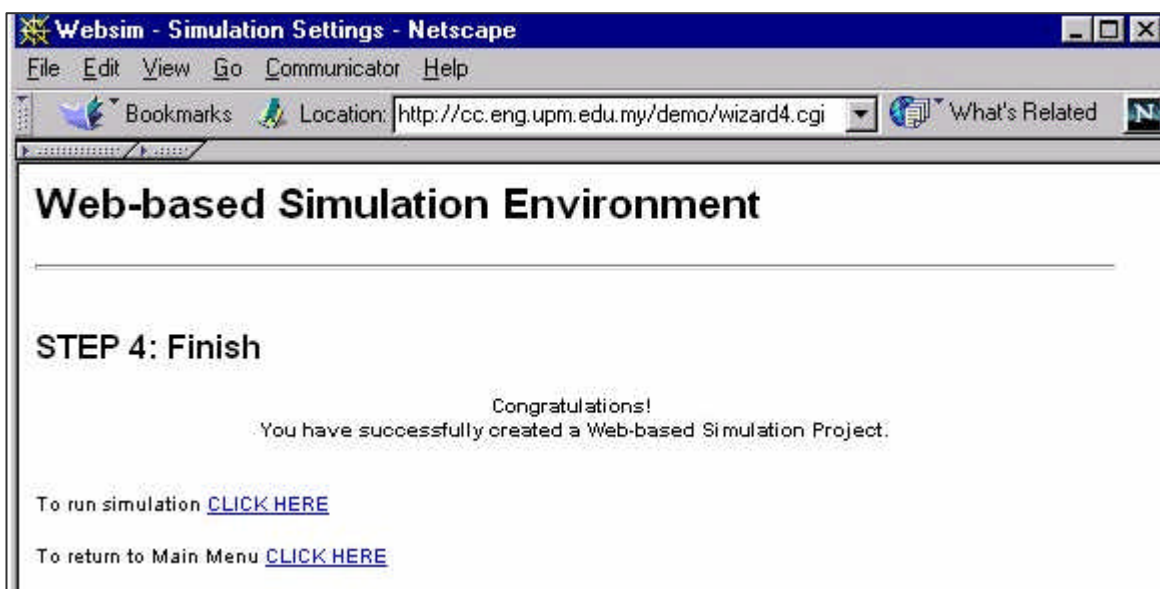


Figure 26: Display URL of Simulation Project Folder to User

Performing Simulation over the Web

Among the central components of Websim are the interfaces to simulation program. Simulation program here refers to a self-developed simulation application in C/C++ uploaded by the Websim administrator. Figure 27 shows how the Common Gateway Interface (CGI), can be used to support the transfer of data from an HTML-

page across the network and the starting of a CGI-script on a server. The CGI scripts are commonly written in Perl but other programming languages could be used too.

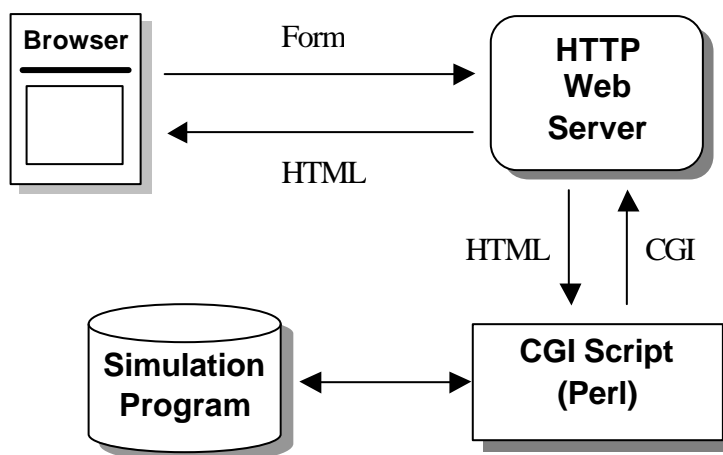


Figure 27: CGI-based Web Connection for a Simulator

It is possible to use a simulation program with the CGI-interface for simulation on the Web if the simulation program supports two properties:

- (i) It must be possible to start the simulation program from a command line. Perl scripts cannot "simulate" the "pushing of a button" in a GUI. The simulator must support command lines like:

```
system (sim_model.exe param1 param2)
```

- (ii) The simulator has to be able to accept the simulation model and simulation parameters as a file and has to be able to write simulation results into a file, which the CGI interface can return to a web-page.

If the simulation program supports the two properties mentioned above, then the user may input simulation parameters into a form container on an HTML page, call a CGI script and execute the simulation program. The following shows the steps in

performing the simulation over the web, referring to the tasks performed by the "sim.cgi" CGI script in Websim.

Step 1: Enter simulation parameters

The first step in performing simulation over the web in Websim is to enter some simulation parameters into a form on web-interface page (as shown in Figure 28). There are two categories of simulation parameters: simulation program input parameters and graph setting parameters. The simulation program input parameters are created by Websim administrator (as discussed in previous section). Example of the graph setting parameters are graph size, graph title, plot styles.

The screenshot shows a Netscape browser window titled "Websim: Fourier Series - Netscape". The address bar shows the URL "http://java.eng.upm.edu.my/websimdemo/project/fourier/". The main content area has a heading "Simulation Input" and three input fields: "Amplitude of wave (default 1.0)" with value "1", "n, number of pi (default 2.0)" with value "2", and "No. of points (default 0.05)" with value "0.05". Below this is an "Output" section with a radio button for "Truncated Fourier Series". Underneath, there are controls for "Plot Type" (radio buttons for "2D" and "3D", with "2D" selected) and "Plot Styles" (a dropdown menu showing "lines"). At the bottom, there are "General Set Plot Size" fields for "X scale" and "Y scale", both set to "300", and a "Border" control with radio buttons for "On" and "Off", with "On" selected.

Figure 28: Input simulation parameters via web-interface page

The web-interface file provides some INPUT and TEXTAREA containers.

For example:

```
<INPUT NAME="param1" VALUE="1" SIZE="8">
<INPUT NAME="param2" VALUE="2" SIZE="8">
<INPUT NAME="param3" VALUE="0.05" SIZE="8">
```

describes a data input field with a default value of zero and a size of 8 characters. If the user writes the number 333 in this field and presses the start button the CGI interface transfers the data to the server in the form

```
name=value&name=value ...
```

or in our example

```
param1=1, param2=2, param3=0.05
```

Then the script on the server is started to execute the simulation program. The script takes the input data as the input parameter for the simulation program. The action attribute in the form tag tells the server which script has to be used:

```
<FORM METHOD=POST
ACTION="http://java.eng.upm.edu.my/Websimdemo/project/fourier/
sim.cgi">
```

Step 2: Invoking simulation program

Once the user click on the "Run Simulation" button, the web server will invoke the *sim.cgi* CGI script. The first task of the script is to call the *invoke_simlib* subroutine.

```
if ($q->param('simulate') eq " Run Simulation ") {
    #Invoke simulation program
    &invoke_simlib();
    ...
}
```

The script firstly will call the *invoke_simlib* subroutine, invoking the simulation program, with param1, param2 and param3 as the command line

arguments. Recall that the simulation program is written in C/C++ which takes a variable number of arguments. One example of the simulation program invoking statement in the *invoke_simlib* subroutine is as follows:

```
system (".../fourier.exe param1 param2 param3");
```

The simulation program will read the values of the three parameters and pass them to the simulation model for processing. After the simulation work is done, the simulation program will write the results to file "output.txt". This is defined in the simulation program codes. The simulation program may have more than one output file.

Step 3: Generating simulation results and graph settings files

The simulation program generates outputs and stores them in a text file. The default simulation output file name is "output.txt". The format of the output text file must be compatible with requirement of *gnuplot* plotting program. Besides, there is another important file named "*sim.dat*", which is used to store commands and settings that we send to *gnuplot* to execute. The settings are obtained from the input parameters entered by the users. This function is implemented by the *savefile* subroutine. The *savefile* subroutine creates a data file "*sim.dat*", consisting of commands and settings that we send to *gnuplot* to execute.

```
sub savefile {
    ..
    $datafile = join ("", $cookie{'dir'}, "\\sim.dat");
    ..
    #Save the settings as file 'sim.dat'
    open (FILE, ">$datafile" );
    #Read graph setting parameter from user
    if ($q->param('plot_type') eq "2d") {
        $ptype = "plot";
```

```

    } else {
        $ptype = "splot"; }
    ..
    print FILE $ptype, " '", $plotfile, "' with ", $pstyle;
    close (FILE);
}

```

The file specified by the variable *\$datafile* is opened for output. All the graph settings parameters entered by the user are processed, arranged and written to the file, which must be of *gnuplot* compatible. The second last statement of the subroutine shows how *gnuplot* should plot (2D, 3D, lines, bar, etc.) the simulation results, specified by the variable *\$plotfile*.

Presentation of Simulation Results

Now it is possible to generate the resulting HTML page with the simulation results embedded together. It will be sent to the standard output, from which it is sent back by the CGI interface to the WWW client. Graphical representation of the results can be shown using *Gnuplot* or other external plotting programs.

First we shall explain the use of process id \$\$ to obtain a unique name for the output graph.

```

#Use process ID as temporary file name
$process_id = abs($$);
$output_gif = join ("",$process_id,".gif");

```

These variables are used to store the temporary files. The \$\$ variable refers to the number of the process running the CGI script, as it does in a shell script. The number is used to create a unique filename even if multiple instance of the scripts run. The

use of the process number for this purpose is a trick that shell programmers have used for decades.

The generation of simulation result as a graphical file is done by using a third party program called Gnuplot. Using existing tools significantly reduce development time because complex graphics capabilities do not need to be reinvented or reprogrammed. One important feature that Gnuplot provides is batch execution, meaning that the program can be executed by typing certain commands at system prompt. The *\$plot_file* variable specifies the gnuplot command with the full path of the simulation data file.

```
$plot_file = join("", "wgnuplot ", $cookie{'dir'}, "\\sim.dat");  
system $plot_file;
```

The *system* command executes the content of *\$plot_file*. The Gnuplot program execute the "*sim.dat*" data file and translates the ASCII input text file generated by the simulation program into a GIF image file. The CGI scripts then return an HTML page containing the generated image. Figure 29 shows a snapshot of the final HTML page containing the output graph.

```
print "</TD><TD><IMG SRC=\"", $output_gif, "\"BORDER=1></TD>";
```

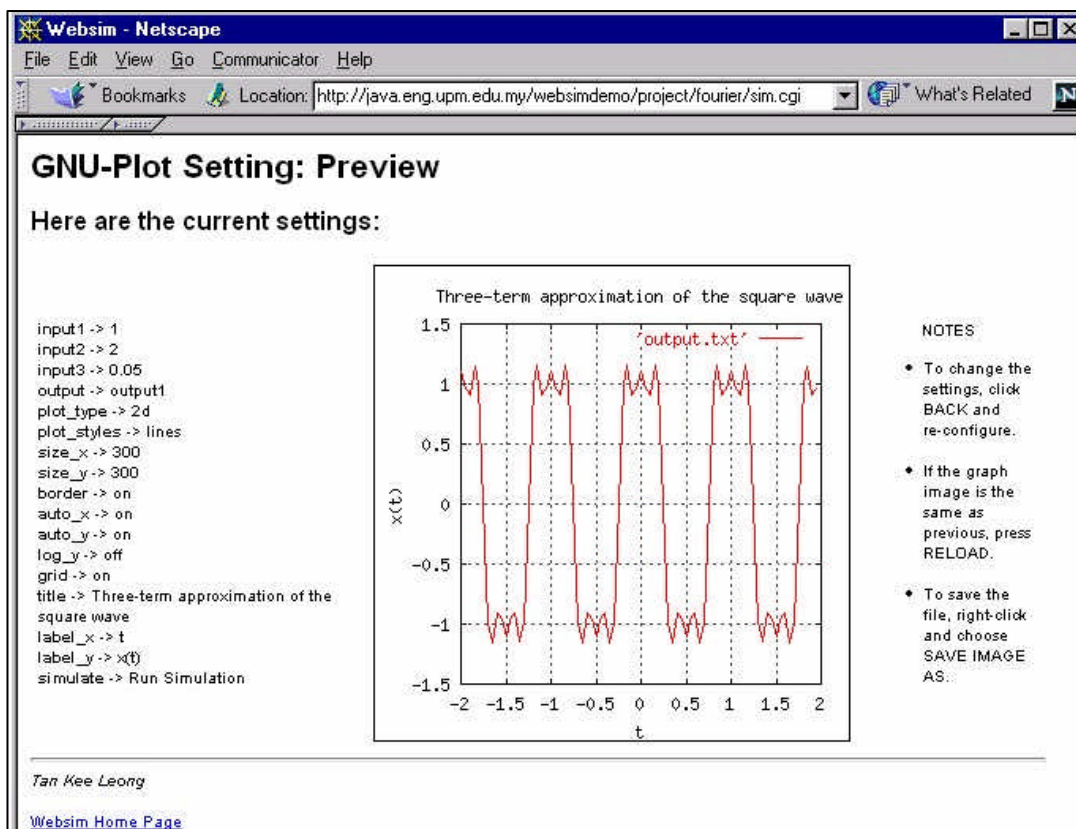


Figure 29: Final HTML page containing the output graph

Security Issues

The Common Gateway Interface (CGI) specification defines a way for World Wide Web servers to interact with query engines and information gateways. However, CGI represents a powerful feature of many Web browsers. This feature allows a Web server not only to provide information, but also to provide access to the computing power of the server. It is important to note that a Web server that supports CGI gateway engines also gives Web browser users a degree of control over what the Web server does. Careful use of CGI can deliver interactive Web sites, user-friendly information retrieval, and access to information not designed for the World Wide Web. But careless use of CGI can and will compromise the security of

the information provider. A CGI application implemented without due regard to security issues will allow the Web browser user much more control over the Web server than the programmer intended. If an organization is complacent on the security of its World Wide Web server, it should expect abuse of its computing facilities, downtime due to malicious attacks, and loss of information integrity of confidentiality. [Eugene, 1996]

We refer mainly from the security measures suggested by Lincoln Stein, who maintained the very famous 'Lincoln Stein's WWW Security FAQ' website [Stein, 2000b]. Lincoln mentioned three unsafe practices that we should avoid when dealing with CGI scripts:

(i) **Avoid giving out too much information about our site and server host.**

Although they can be used to create neat effects, scripts that leak system information are to be avoided. For example, the "finger" command gives would-be intruders valuable information on what daemons are running on your system.

(ii) **When coding in a compiled language like C, size of user input must be limited.**

A MAJOR source of security holes has been coding practices that allowed character buffers to overflow when reading in user input. The buffer overflows and crashes the program; in some circumstances the crash can be exploited by the hacker to execute commands remotely. Of course, once we read in the data, you should continue to make sure your buffers do not overflow.

(iii) **Never pass unchecked remote user input to a shell command.**

In Perl this includes `system()`, `exec()`, and piped `open()` functions as well as the `eval()` function for invoking the Perl interpreter itself. In the various shells, this includes the `exec` and `eval` commands. Backtick quotes, available in shell interpreters and Perl for capturing the output of programs as text strings, are also dangerous.

We also refer the security precautions implemented in [Morton, 1999] and [Dincer and Fox, 1997]. Just like Websim, the server was configured in a very careful manner in order not to hinder the security of the entire system.

In this thesis, we consider security as a very important issue because of several server-side services provided by Websim, among them are:

- Availability of file upload features.
- Files to be written to the server.
- Execution of binary on the server.

The discussion of a comprehensive analysis of the security risks is beyond the scope of this project, but our approach here is to anticipate potential attacks and implement several strong security measures. Here we discuss the security precautions applied in Websim, which are divided into two levels of implementations: the web server and CGI scripts.

Web Server Security

As a first precaution, web server authorization mechanism is used to restrict the accesses to the system. The web authorization mechanism allows the administrator to password protect the Websim root folder, where all the scripts are installed. This is a very important security measure, which can only be implemented by the administrator who first set up Websim on the web server. This makes it easier to use the NTFS system file protection mechanisms to protect directories from other users of the same system. In the common nobody approach, the access rights for all the files should be set as readable and executable by the world, which make them vulnerable to bad guys trying to steal homework solutions of other users.

Secondly, the web server is configured to not generate dynamically produced indexes but return an error message instead. This is to prevent the entire contents of the CGI directory to be displayed to the web user. Certain files may contain important security information such as path and configuration information, which in the hands of the hacker could bring much damage to our system.

CGI Script Security

Beyond the fact that web servers are insecure to begin with, web servers make a bad situation worse by allowing users to take advantage of CGI scripts. [Sol, 2000]. There are 4 security measures implemented in the CGI scripts.

- (i) First of all, Websim will only browse one level directory, which it assumes is the top web directory for the site. There is no way for the users of the system to go out of their directories by using any of the utilities provided by the system. Commands such as delete/rename/edit file always manipulate only

items in the respective simulation project folder. It will also browse only text, html and CGI files and not other file format. This restriction prevents Websim from being able to access any files outside the site where we install Websim.

- (ii) Websim allows administrator user to save files in directories on the server. Thus, access to Websim must be limited to those who have write permission on those directories. At our installation, we achieve this security by requiring the user to produce a user name and password when Websim is first started in a browser. This is achieved by placing an access file (called `passwd.txt`) in the directory from which Websim runs; the access file specifies which users may log on as administrator. A password file is generate containing the usernames and associated passwords of all the valid users of the system. All the documents directories and CGI script directories are protected. A user who does not supply a matching username/password pair is not allowed to access the project directories.
- (iii) As for the third security measure, the input fields typed by the users are checked against all kinds of special meta-characters, and rejected if determined to be invalid. The only text data it accepts are file names and it restricts file names to alphanumeric characters plus `_ - ~ / .` but no whitespace. In a sense, the users of the system have fewer rights than they would have with an actual account on the same system. A users also can only run the executables (simulation program) on the particular simulation project folder.
- (iv) Finally, Websim will accept form data only from itself, by checking the `HTTP_REFERER` variable (it generates its own web forms). If the referring form is not itself, it will not process any form data submitted and will return

only the default web page indexing the top of the site (the parent directory of Websim's directory). This helps to prevent arbitrary form data from being submitted, either with method POST through an alien form or with method GET and the query string or path info.

*"All data is fraudulent. All communications are attempted hacks.
All clients are thieves. Technology is only my first line of defense"
- morning litany for a Web Server Administrator (Selena Sol)*

CHAPTER V

RESULTS AND DISCUSSION

Introduction

This chapter describes the experiments conducted to test, verify, analyse and evaluate Websim. We also explain several Websim limitations and the proposed solutions. The areas that have been evaluated are:

- (i) Functionality and overhead of Websim.
- (ii) Performance measurements of Websim using three simulation project samples.

The main objective of the tests is to verify that Websim is able to function according to the design and is sufficiently flexible to be implemented in a real network environment. We will study Websim performance in terms of its simulation program execution time and upload time.

Experimental Setup

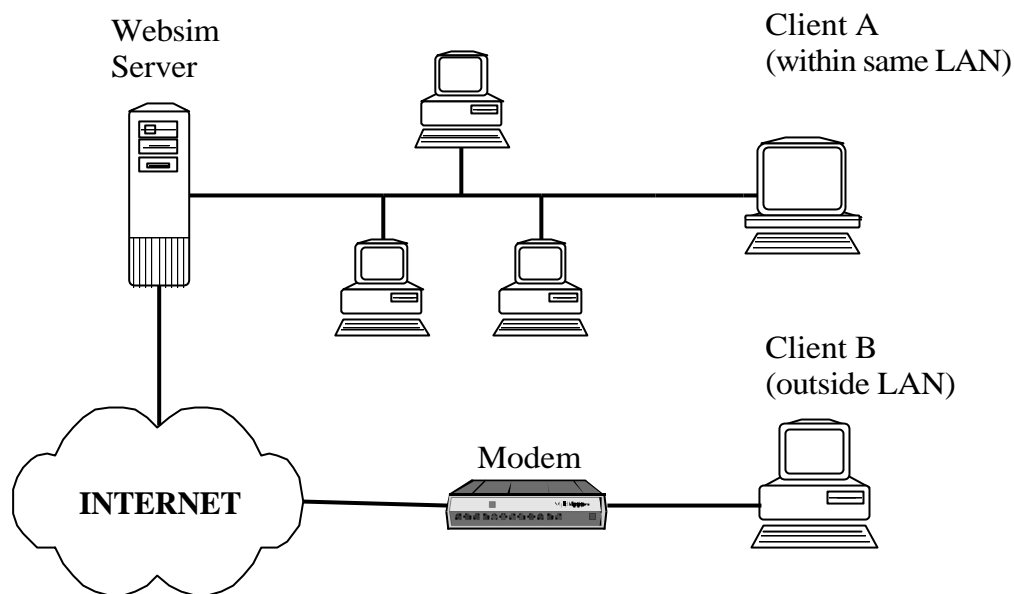


Figure 30: Websim Experimental Setup Environment

Websim test setup environment is shown in Figure 30. It consists of a server and two clients (A and B). Websim server is the most important component as this is where all Websim applications were installed. Client A is a workstation located in an adjacent room to the server, and it is connected to the Websim server via Ethernet LAN with transmission capacity of 10 Mbps. Meanwhile, client B which is an ordinary home-user computer, was located outside the campus LAN. To access Websim, it used a modem dial-up connection to the Internet with transmission capacity of 56 kbps for download and 33.6 for upload.

Websim was setup in a server computer, which has a HTTP server running and has Perl interpreter installed. All the Websim program and script files were stored in a web-enabled directory according to its original arrangement, as discussed in Chapter III (page 48). Next we shall describe the configurations of all the three computer machines.

Machine Configuration

All the machines used for this experiment are based on Intel Pentium series processors, equipped with the basic hardware specifications and each has a 10 Mbps network card. The OS is Microsoft Windows operating system and has either Netscape Navigator and/or Internet Explorer web browser installed. Since Navigator is part of our development testing tools, it is our preferred choice. Table 3 lists the configurations for each machine:

Table 3: Configurations for Websim Server

Computer Name	Sun
IP	202.184.18.11
DNS	java.eng.upm.edu.my
CPU	Intel Pentium II (450 MHz)
RAM	64 MB
OS	Windows NT Server 4
Web Server	Omnihttpd version 2.02
Perl Interpreter	ActivePerl (Build 519)
Web Browser	Netscape Navigator 4.7

The web server and Perl interpreter can be downloaded from the OmniHTTPd homepage at <http://www.omnicron.ab.ca/httpd/> and the Activeperl homepage at <http://www.ActiveState.com/ActivePerl/> respectively.

Table 4: Configurations for Client A

Computer Name	Hpkayak800
IP	Dynamically assigned.
CPU	Intel Pentium III (450 MHz)
RAM	128 MB
OS	Windows NT Workstation 5
Web Browser	Netscape Navigator 4.73

Table 5: Configurations for Client B

Computer Name	<i>none</i>
IP	Dynamically assigned
CPU	Cyrix 166 MHz.
RAM	48 MB.
Modem	56.6 kbps external modem.
OS	Windows 95
Web Browser	Netscape Navigator 4.5

Functional Testing

This section describes the sets of tests on the functionality of Websim. This is to ensure and verify that all Websim services (as discussed in Service Section, page 34) operates and performs as designed. The functionality of Websim was tested using three different simulation modules:

- (i) A Diode circuit
- (ii) Fourier Series Simulation
- (iii) Go Back N

A C/C++ simulation program was developed for each of the module. The full source codes of the three programs are shown in Appendix A-1, A-2 and A-3. The program is then compiled and turned into a binary file. Finally, each binary file was uploaded to Websim server, and stored in different Websim folders.

The first two simulation modules show how Websim could be used as computer assisted learning tool in the education sector; while the third simulation project extended the functionality of Websim to act as an online simulation library.

An online simulation library allows for a simulation program to be uploaded, and could be accessed over the Internet. Next we shall give brief information on each of the simulation program:

(i) **A Diode Circuit**

This simulation module is used to generate and show the characteristic of a semiconductor circuit. It proves that for any negative voltage, a very small negative current will flow, while for positive voltages the current will be positive and increase rapidly with v . We use the `range1` and `range2` variables to represent the voltage domains, where `range1` represent the minimum voltage, and `range2` represent the maximum voltage. By using different values for `range1` and `range2`, we could see from the graph of Websim plot, that the diode is practically Off for negative voltage values, and On for positive ones. In other words, the diode acts as a rectifier. This sample is taken from [Breiner and Biran, 1999]. An example of the output graph by Websim is shown in Figure 31.

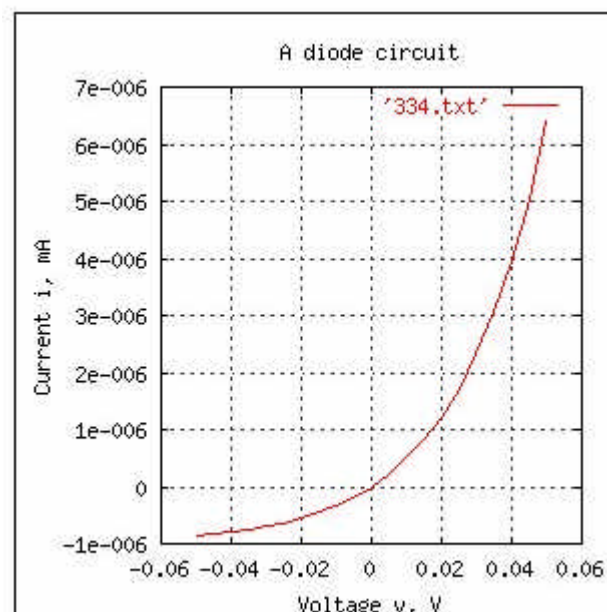


Figure 31: Characteristic of semiconductor diode

(ii) Fourier Series Simulation

The second program generates a truncated Fourier series, when given: (a) amplitude of wave; (b) angular frequency; and (c) the increment points. By inserting the three inputs, Websim will generate an output graph showing the three terms approximation to the square wave. The graph obtained is shown in Figure 32. The initial output graph that Websim generates might look rather 'broken', and the resolution can be improved by having a smaller increment points. This sample is also taken from [Breiner and Biran, 1999].

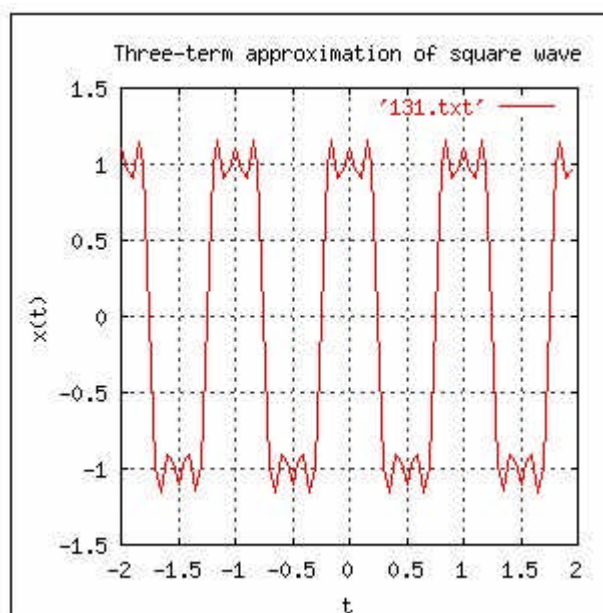


Figure 32: A truncated Fourier series

(iii) Go Back N

The third simulation module is used to find the effect of timeout on GBN throughput given: (a) percentage of packet losses; (b) percentage of check-sum error; and (c) timeout range. The result from Websim graph (Figure 33) shows the effect of timeout on Go Back N throughput for a packet size of 54 bytes. This simulation program has been adapted and modified from [Sliman, 2000].

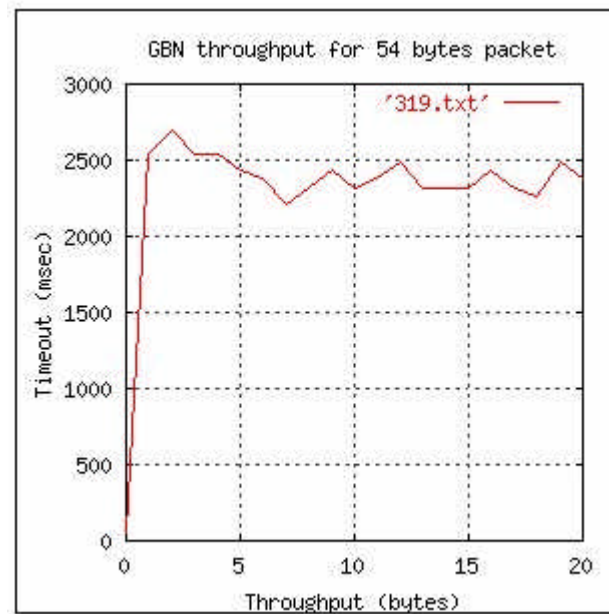


Figure 33: Simulation output showing the GBN throughput for 54 bytes packet.

Performance Evaluation

This section discusses the performance of Websim by studying the time taken to:

- (i) Upload the simulation program from client machine to the Websim server; and
- (ii) Execute simulation program and return the output graph.

The simulation program refers to the binary executable file as discussed in previous section. But since the size of the binary file for the Diode (112KB) and Fourier (103 KB) simulation program are almost the same, a larger binary file named 'heavy.exe' (232 KB) is used for the purpose of testing and evaluation.

The performance is evaluated under two scenarios:

- (i) Based on the geographical distance between Websim server and client machines.

- (ii) Based on the number of concurrent users accessing the Websim server.

The results are presented next in the following tables. Table 6 shows the evaluation results for Websim when the access is done at the same server machine.

Table 6: Websim performance when accessed from local machine

Simulation Program		Upload Time (s)			Execution time (s)		
Name	Size (KB)	1 ss.	2 ss.	5 ss	1 ss	2 ss	5 ss
Diode.exe	112	0.30	0.30	0.34	0.20	0.20	0.23
Fourier.exe	103				0.19	0.21	0.22
GBN.exe	70	0.29	0.30	0.31	0.22	0.23	0.26
Heavy.exe	232	0.32	0.32	0.45			

ss = session(s)

Table 6 shows that the upload and execution time are very insignificant. This is because the Websim client accessing its services is located at the same machine, thus all the processes are handled internally. The result obtained from this scenario is regarded as the ideal result.

Table 7: Websim performance when accessed from a machine within the same LAN

Simulation Program		Upload Time (s)			Execution time (s)		
Name	Size (KB)	1 ss	2 ss	5 ss	1 ss	2 ss	5 ss
Diode.exe	112	0.31	0.32	0.34	0.20	0.21	0.22
Fourier.exe	103				0.19	0.22	0.22
GBN.exe	70	0.29	0.29	0.30	0.2	0.21	0.22
Heavy.exe	232	0.30	0.33	0.37			

*ss = session(s)

Table 7 shows the performance when this simulation is done from another machine – Client A, located within the same LAN as the Websim server. We notice

that the results obtained from Table 2 are very much similar to those in Table 1. This similarity is due to the availability of 10Mbps bandwidth within the Ethernet LAN. This amount of bandwidth is sufficient to provide an almost ‘instant access’ between client A and Websim server.

Table 8: Websim performance when accessed from another machine outside campus network, via Internet.

Simulation Program		Upload Time (s)			Execution time (s)		
Name	Size (KB)	1 ss	2 ss	5 ss	1 ss	2 ss	5 ss
Diode.exe	112	23	51	178	5	6	15
Fourier.exe	103				4	5	9
GBN.exe	70	18	33	86	4	5	11
Heavy.exe	232	47	95	302			

*ss = session(s)

In comparison to the previous two tables, the upload and execution time shown in Table 8 are much longer. One major factor contributing to this delay is the limited bandwidth of our PSTN, where its maximum download bandwidth is 56.6kbps and 33.6 kbps for upload. But due to the large number of Internet users, usually the average bandwidth available for each user is approximately 5 kbps only. Another contributing factor for this poor result is the limited availability of system resources such as amount of free memory and CPU utilization at the client C machine.

Limitations and Proposed Solutions

Since this thesis is a pioneer work, its original aim was just to develop a prototype of web based simulation environment, combining CGI and Javascript technologies to demonstrate its idea. Even though the main aims and its more

specific objectives have been achieved, the final Websim prototype still require a further research. It will be ready for open public usage once the following two crucial limitations are overcome: Websim security and multi-user support.

Websim security

Since Websim can write, execute and upload files to the server, security is a very important issue. Even though we can provide security measures to prevent unauthorized users or data from entering the system, Websim by default allows the upload of binary executable file to the web server. Binary file format is chosen as the simulation program file format because of its very fast execution time and compact size. But it also prevents our CGI script to open and look at its content. A binary file uploaded may contain some harmful system commands such as copy, delete, format, etc. which may damage the Websim server. This is perhaps the most challenging problem and major limitation of current version of Websim.

To handle this problem, we would like to propose a system level mechanism where a security “wrapper” application is put around the binary executable file. When a binary file is first uploaded to the server, the wrapper application will encapsulate the file, and check for the system commands to be executed by the binary file. This can be achieved by using the wrapper application to load Websim into a certain protected area of the system available memory. The application will be executed at the protected memory space and its activities monitored by the wrapper application. If any of the harmful commands such as copy, delete or format is detected, the wrapper application will automatically delete the binary file.

A second solution proposed is to develop a special purpose Perl module, containing a special function that allows the safe execution of binary file. Unlike Perl

commands such as `open`, `system` or `exec`, the module shall restricts the operation of the binary file by allowing a minimum write/delete capability and disallow it from referring to variable outside the working directory.

Multi-users supports

Websim allow many users to access and perform simulation simultaneously. Hence, in order to have a fair and optimum support to all users, a CGI classical method have been implemented. We used variables based on the process ID – (`$$` in Perl) to create temporary files for each request. The `$$` variable refers to the number of the process running this program, as it does in a UNIX shell script. The number obtained from `$$` is used to create a unique filename, even if multiple instances of the program run. The process identification is prefixed to each filename. We foresee that this method will not be able to handle large amount of requests and complex simulation programs which demand high CPU utilization.

As a solution to this potential problem, we propose the use of multi-threading. A thread is placeholder information associated with a single use of a program that can handle multiple concurrent users. Thread makes it easier to implement simulation by using the process interaction approach, where each thread represents a process in the simulation environment. If multiple users are using the same program or concurrent requests from other programs occur, a thread is created and maintained for each of them. The thread allows a program to know which user is being served as the program alternately gets re-entered on behalf of different users. Most of today's operating systems provide support for both multitasking and multithreading. They also allow multithreading within program processes so that the

system is saved the overhead of creating a new process for each thread [whatis, 2000].

However, at the moment of this thesis writing, the thread module provided in the latest version of Activeperl (version 5.6) is still an experimental module. Users are told to use at their own risks. According to the release notes, both the interface and implementation are subject to change drastically. Perl 5.6 and later have the beginnings of support for interpreter threads, which (when finished) is expected to be significantly different from what is described in the current version [Activeperl, 2000].

Simulation Program

The simulation program is another significant limitation of this project. Simulation program refers to the executable file created using C/C++ compiler. When writing the simulation program in C/C++, we have to specify a fixed number of input arguments and output the results to a text file. This gives rise to several limitations, such as:

- Number of arguments could not be added/modified, and value of all the input arguments have to be manually supplied. Hence everything is static.
- The results in the text file must follow the standard GNU Plot data files format, such as specifying the first column as x and the second column as y .

A script to test and auto-detect the input/output parameter of the simulation program is proposed. The structure of the simulation program template too, can be improved by creating some common and useful libraries, which can be included easily in the simulation program. Besides the usage of C/C++ as the language for the

simulation model, we may add the choice of the language by supporting other interpreted language such as BASIC and FORTRAN. But this would require the installation of the language interpreter at the server.

Conclusion

The various tests conducted as discussed in this chapter verify the achievement of Websim objectives. The functional testing shows how Websim could be used as teaching tools, as well as an online simulation library. Three simulation project samples were developed, and the tests conducted on them gave a very encouraging and satisfactory result. The result also shows that its performance is best when Websim server and its clients reside within the same LAN. There are many external factors which directly or indirectly contribute to Websim performance, such as number of concurrent sessions, distance between client and server, Websim server available resource (available memory and CPU utilization), network bandwidth, etc.

With the availability of newer technologies and better development tools for Websim in the future, a superior results could be obtained. We further discuss the limitations of this project and future works proposed for next generation of Websim in the following chapter.

CHAPTER VI

CONCLUSION AND FUTURE WORKS

This chapter explains several identified areas where further research can be done, in creating the next generation of Websim, and ends with a conclusion where we describe various contributions of Websim and a summary of this thesis.

Future Works

Various areas of Websim have been identified for future works. Areas which are highlighted for improvement are essentials to enable Websim to work more effectively and efficiently.

Additional Services

This current version of Websim provides five services, which offer some limited features sufficient to performing simulation over the web. There are many features improvement of existing services that can be made. For example the site manager may provide a visual file browsing tool similar to the Windows-based applications that performs additional file operations such as copy, delete, rename,

print, etc. Improvements can also be made to the text editor where we could provide a multicolour and “what you see is what you get” text area display.

Certain new tools can be introduced, such as a time clock to represent the simulated time when the simulation is being carried out. The clock, measured in second or millisecond will change on the screen as the simulation event takes place. Secondly, we suggest Websim to provide a display that gives numerical values statistics summarizing simulation and processor utilization data. The data provided include the percentage of busy, overhead, and idle time, total count, volume of messages sent and received, etc.

Database

We may use a database to store the Websim users information, as well as the simulation results. Access to the database will be implemented using CGI scripts. Its goal is to accelerate a new presentation of older simulation results, and to allow comparisons between results of different simulations runs of the same simulation program. This task can be achieved using a Perl module called Win32::ODBC by Roth Consulting, [Roth, 2000]. Win32::ODBC is a Perl 5 extension that provides access to the Open DataBase Connectivity (ODBC). ODBC is an API that allows a programmer to abstract a program from a database such as Microsoft Access, Sybase and Oracle. ODBC currently exists on the Microsoft Windows platforms as well as the Macintosh and Unix.

Graphical User Interface

Java and Javascript may be used to improve the interactivity and user interface at the client site in addition to static components such as HTML. In the current version of Websim, Javascript is mainly used to validate user's input. But we can extend its usage by using its ability as mentioned in Chapter II (page 22) to provide more dynamism to the HTML pages.

Meanwhile, Java and its library such as the Abstract Window Toolkit (AWT) is an excellent resource for building GUI that we see in many commercial software packages [Java, 2000]. It supports everything from creating buttons, menu, dialog boxes, and can be used to display simple windows containing graph and statistical charts too, as shown in Figure 34.

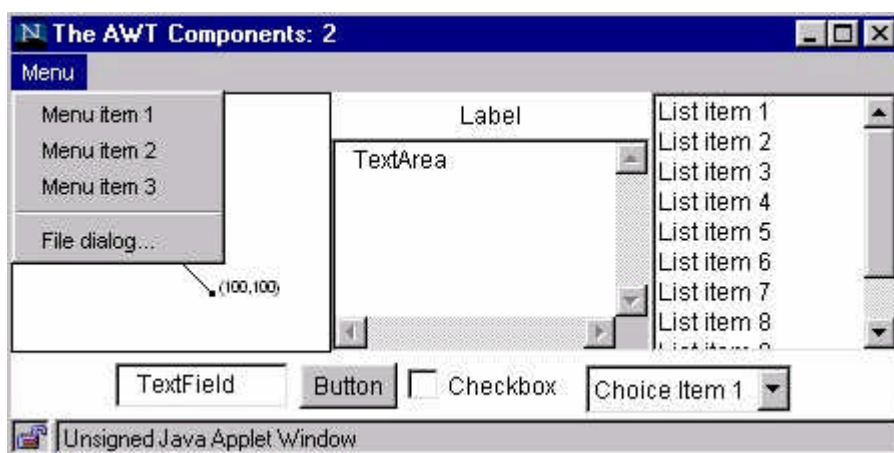


Figure 34: Example of components created using Java AWT class.

A later version of Java provides a more advanced library known as Swing, it has new ability to create more sophisticated windows like components, such as colourful buttons and scroll bars, that are independent of the windowing system for specific operating system [Swing, 2000]. Some of the components that can be created using

Swing is shown in Figure 35. The only limitation of Java is its speed, which would slow down the performance of the system.

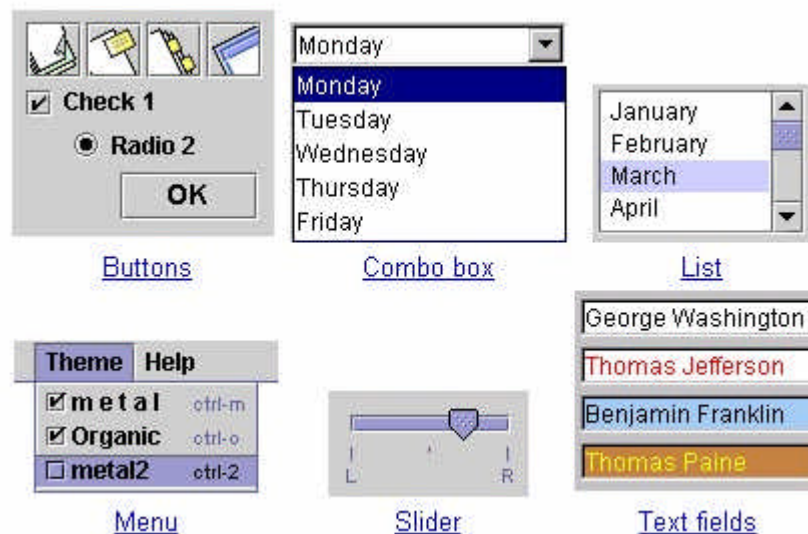


Figure 35: Example of several Java Swing Components

Animation and Visualization

Animation and visualization helps students to better understand the functionality and behaviour of a simulation model. There are two general approaches to implement animation and visualization of the simulation results over the web. The first approach is by using a Java based animation system, similar to the existing Skopeo project [Skopeo, 1997], which utilizes Java's graphic and multithreads capabilities. In this project, the animation system is maintained on a host server. Then, certain important components of the animation system are loaded at the client side as applet. If the user closes the page, all the components at the client's machine are destroyed. This type of animation system is trace driven and needs a layout and trace file to perform an animation.

The second approach is by using the Virtual Reality Modelling Language (VRML). VRML is a language for describing three-dimensional (3-D) image

sequences and possible user interactions with them. Using VRML, we can build a sequence of visual images into Web settings with which a user can interact by viewing, moving, rotating, and otherwise interacting with an apparently 3-D scene. For example, the simulation graphical results document may redirect the user to a VRML file. When the browser recognizes this particular information type, it will launch the VRML plugin, allows us to construct 3-D geometry and have it rendered in real time.

However, the current problem with web-based animation programs and visualization techniques is that they are not parameterised. There is no dynamic binding to the program source code, which is to be visualized or animated. Whenever the source example is modified or expanded, the animating sequence has to be adapted manually. Researches have been carried out by designing a parameterised animation tools, which is able to animate and visualize parts of the program or the source automatically and dynamically, without external construction of visualization sequences.

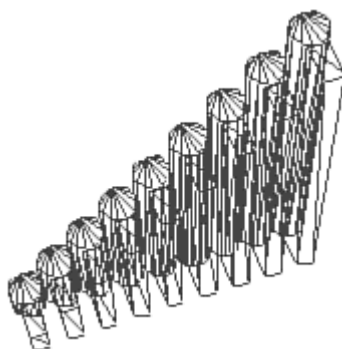


Figure 36: A 3-D image created in VRML

Websim Documentation and User Guide

Documentation of Websim is built during various phases of development. We propose a thorough and comprehensive documentation to be compiled and

maintained for future use. It should be made available online, and kept in cross platform compatible file formats such as HTML, PDF and Postscript. The documentation should contain all the facts and specification of Websim, its help files, FAQ, tutorial, reference and user guide. The user guide would be specifically written for the new user or Administrator of Websim. We also plan to implement version control for Websim. Its goal is to show how one could use Websim as a tool to integrate simulation program with the web. It would also show a variety of features supported by Websim, and example of some basic operations.

Websim Contributions

We describe here three major contributions of Websim, particularly to the research and educational sectors. The first contribution is the ability of Websim to produce a web-based simulator given a standalone simulation program. Websim is able to receive the simulation program in executable format and provide a web interface for it. This gives the flexibility and convenience of using the programming language of choice for the simulation modeller., and to integrate it with the web.

Secondly, Websim allows the storage of simulation program on a web server, thus acting as an online store for simulation programs. This allows the sharing of simulation program over the Internet to an exclusive user group or to the general public. A user can access the simulator via any standard web browser without having to install any software on the user machine. A set of services has been developed for the administrator and users to perform simulation over the Internet.

Thirdly Websim could act as a teaching tool in school and universities. This would be very useful especially for courses involving modelling and simulation. For example lecturers could develop a simulation program related to his course, upload it to Websim, and allow the students to perform the simulation over the Internet, hence they could verify their understanding during lectures. In other words, Websim allows teaching and learning to be done through the Internet, hence assisting students to have a better understanding of certain topics or concepts in their learning.

Conclusion

This thesis reports on the preliminary results of an on going effort to design and develop a web-based simulation environment, to distribute simulation experiments over the Internet. A prototype of that simulation environment called Websim has been developed using CGI and Javascript. In this thesis, the architecture, design, implementation and performance evaluation of Websim have been presented. We believe that the use of WWW, CGI and Javascript as the enabling technologies will provide a breakthrough for web-based simulation. First, the infrastructure provided by the Internet obviates a multiprocessor hardware for simulation users, it is already feasible to distribute simulation models over different hardware platforms through the Internet. Secondly, the use of web browser on the client side makes the application become fully portable and avoid steep learning curve.

Future projects in the domain of web-based simulation environments have to solve the open problems along with the research, designing and implementing additionally required components of such an environment. The usage and acceptance

of the presented system for commercial use is uncertain. For this application of web-based simulation environment, questions about security and copyrights of the simulation program are especially important, and need serious consideration.