Bringing Software Tools to the Web — Architecture Issues and Porting Experience

Michael R. Lyu

Computer Science & Engineering Dept.
The Chinese University of Hong Kong
Shatin, Hong Kong
lyu@cse.cuhk.edu.hk

J. Schoenwaelder

Operating Systems & Computer Networks Dept.
Technical University Braunschweig
Braunschweig, Germany
schoenw@ibr.cs.tu-bs.de

Abstract

Due to the popularity of the World Wide Web, many users demand that software tools be brought up and made available in the Web. There are challenging problems we have to solve to meet this demand. This paper discusses the architectural issues and technical decisions involved in porting a stand-alone software tool so that it can run on the Web interactively. We examine the general architectures required for this set-up, and present our experience in bringing a stand-alone tool into the presence of the Web. In particular, we describe the procedure to bring a computation-intensive software reliability analysis tool, CASRE, to execute on the Web. The resulting software, called Web-CASRE, takes advantage of the Tcl/Tk plugin facility to download a Tcl script for its front end user interface, while communicating with its back end executables. The design decisions regarding communications protocol, security policy, data storage, and consistent computation environment are also discussed in this paper.

Key words: Web Technologies, Client-Server Application, Tcl/Tk Plugin, Software Reliability Tool.

1 Introduction

Since its first official appearance in 1993, the World Wide Web (or simply the Web) has drawn tremendous attention among not only academic professionals, but also computer illiterates [2]. The Web's phenomenal popularity stems from the way in which it enables individuals to navigate their own paths through the abundant information repository provided by any other individuals or cooperations. People share ideas, hold discussions, discover theories, and learn new techniques freely and openly in the Web. The Web is also highly interactive, making everyone potential desk top publisher, forum partitioner, and information explorer. The Web is designed to be an open-ended multimedia system for the delivery of both text and non-text based information. The flexibility of the Web has led to a tremendous growth of interest in the potential of network-based information systems. Combining with large user base, this technology has generated tremendous impact for innovative methods of transacting business for companies, and worldwide opportunities for communications amongst individuals.

The Web is based on standard networking concepts and has a client-server architecture. Each web site runs a server program that responds to requests for information resources from browser programs, the clients. Each request and its corresponding response form a discrete transaction. Browsers may make requests to different servers depending on the links followed by the user. The servers do not keep track of the origin of requests, and regard each request as a completely new one even if it comes from the same browser. Namely, all the requests are assumed independent of each other. Where a document is composed of a series of information resource (for example text with embedded pictures), a separate request is issued to the server to retrieve each of the resources.

Currently the only non-text content it has been possible to incorporate into a web document has been the static bit-mapped image. However, documents on personal computer systems have evolved complex multimedia capabilities: the distinction between a word-processor document, spreadsheet and drawing are becoming blurred and a variety of systems are evolving in which documents are merely containers for a wide range of related multimedia information objects. Much excitement is being caused by the possibility of extending these facilities to the Web and being able to retrieve documents which contain windows showing live video, a section of a spreadsheet, or perhaps even the user interface of an application. Users may be able to interact with the content of the document, and perhaps even update it collaboratively with others.

The explosive growth of the Web as platform for computing services introduces the need and desire for system tool developers to provide their software over the Web as well. The advantage of executing a software tool directly from the Web is three-fold: timeliness, portability, and maintainability. Users will expect more software to be instantly accessible through their web browser. Furthermore, the absence of setting environment variables or going through explicit down loading and installing procedures greatly encourages the portability of a software tool. Finally, the software technology associated with the Web offers a simplified yet standard execution environment for software tools, using popular web browsers and user downloading mobile code in the front end, while connecting web servers for language-dependent executables in the back end. This avoids the intensive tool supporting work such as porting to user-requested platforms and shipping new versions. The potential of this client-server paradigm allows a tool's computationally intensive solution algorithm to run on special-purpose server-based hardware that may not be readily available at ordinary user sites.

Several experimental Web-embedded applications and specialized HTTP servers supporting Web-embedded applications have appeared in the literature [DM97, RR96, TBW96]. In this paper we discuss Web-based client-server solutions for computation-intensive software tools. To make such tools presentable over the Web, we split them into a front end GUI and back end model "interpreter" and solver engine. The GUI is made available on a web server, and users can start a session by simply downloading the GUI in a web page through an HTTP. The solver (and computationally intensive modules such as numerical algorithm for reliability modeling) remains on the server, and can be accessed throughout the user session, as described by the users. We address the architecture issues involved in this approach, and discuss the consequences of the Web-based solution on the design of such tools for its interacting GUIs, model interpreters and solvers. We also present a case study for the porting of a computation-intensive software reliability tool over the Web.

2 Client-Server Architectures for Web Tools

A computation-intensive software tool is typically implemented in two major parts: its computation routines (the solvers or "executables") and its user interface ("the GUI"). Traditional software tools are designed such that these two parts are mingled together. To make a tool available in the web, these two parts have to be separated: the user interface runs on the client side, while the executables run on the server side. In addition to its natural fit to the Web environment, the client-server architecture offers the advantage that the client can run the GUI of a tool efficiently in a local cache, while computation-intensive solver tasks can be located at the server for background computation, not withholding local resource in the client side.

Like a surgery, separating the originally intertwined GUI code and solver code could be a delicate and tedious task requiring careful planning and clean interface cutting. Decisions have to be made regarding which part of the code should be ported to the client side, and which part should remain in the server side, as well as how these two sides communicate with each other. In the following we describe a sequence of steps to bring a software tool to the Web.

Figure 1 shows a typical stand-alone software tool for its GUI and its executables. The GUI is what the user of the tool will see and operate, while the executables, running on the background, receive commands from the GUI, perform the required computations, and send the results back to the user through the front end GUI. The GUI and the executables of a

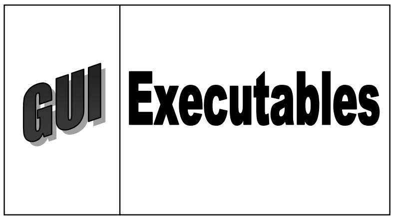

Figure 1: Traditional Software Tool Set-Up

tool are usually implemented in different types of languages: GUI uses graphics-enhanced languages like X Window [6], Windows Development Toolkit, Motif, and Tcl/Tk [4], while executables are coded in computation-intensive languages like C, C++, Fortran, and other traditional programming languages. GUI and executables are usually bundled together due to their high interactions. They share the same global environment, and can transfer data between them conveniently.

In order to bring a stand-alone tool to the web, the GUI and the executables are separated, as seen in Figure 2. After the separation, the GUI is running on a local site (the client site), and the executables are running on a remote site (the server site). A communications link connects the client and the server. At each end of the communications link is the "hook" from each side to define the control and data exchange protocols. The original interface between the GUI and the executables is shrunk into the hooks which link both sides.

Since the GUI and executables in the original software are mutually involved, the hook in each side turns out to be a complicated layer. The originally bundled GUI-executables pair share the same computation environment. This environment is no longer available, and these

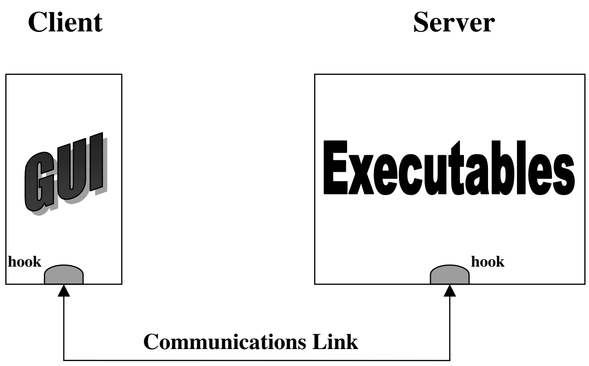

Figure 2: Client-Server Software Tool Set-Up

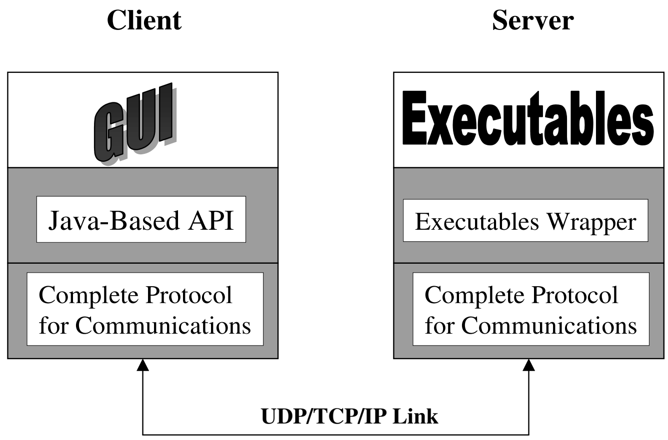

Figure 3: A General Architecture for Web-Based Tool Set-Up

two parts will have to completely rely on their hooks to exchange messages. Consequently, the general architecture for this web-presentable tool can be described in Figure 3.

Figure 3 shows that the GUI on the client site will run on top of a Java-based API (due to Java's portability), which defines the required operations for the GUI to communicate with its back end, the executables. The executables have to be wrapped properly by well-defined command execution and data accessing mechanisms. A complete communications protocol has to be clearly specified between these two sides to pack and transfer data for various kinds of operations, using, for example, UDP/TCP/IP links.

While the software architecture in Figure 3 represents a general solution to the creation of web-based tools, it may be too aggressive. Although Java is popular for its portability and web-readiness, it could be very tedious to implement all the required APIs for the tool's GUI functions. Moreover, the communications protocol is hard to be completely defined. Consequently, the porting task of existing software tools to the Web may result in a complete re-implementation of the GUI in Java.

In our efforts to bring stand-alone software tools to the Web, we simplify the set-up of

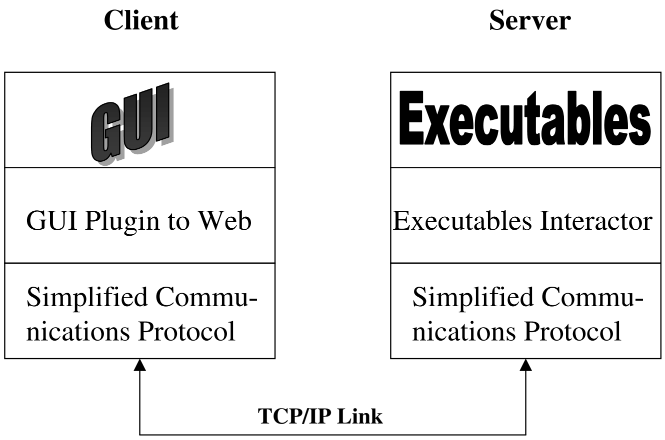

Figure 4: A Simplified Web-Based Software Tool Set-Up

the environment as shown in Figure 4. In this set-up we limit the target tools to those whose GUIs have existing plugins to particular web browsers. We will further simplify the communications protocols so that all the data items are packed as ASCII streams for easy up-loading and down-loading, and the executables interactor ensures a synchronized environment between the front end and the back end.

There are several issues that we have to address for such web-based tools:

1. Storage place of the data.
   The storage of the data is a main concern in the client-server environment. The data can be stored either in the client side or in the server side. If the data is stored in the client side, it may be very time consuming to transfer data back and forth for computation purpose. On the other hand, if the data is stored in the server side, it may violate privacy and propriety requirement for most users.

2. Security concern.
   When the client downloads executable user interface from the server, there may exist

security holes if the server is not trustable. When all the commands of the GUI code are allowed to execute, malicious action can easily invade the system. However, if no commands are allowed, the GUI code becomes useless for sophisticated functions.

3. Interface simplification.
   In order to limit the expensive communications between GUI and executables, the interface between them should be simplified. Small and frequent interactions should be limited, if possible, and replaced by larger interactions with lower frequency.

4. Front end computation.
   The purpose of running executables in the server side is to perform complex computations in a more powerful machine without having to download large server code. This is done at the expense of data communications. Some simple computations, when implementable in the GUI language, can be performed in the client side to avoid the communications cost.

5. Communications.
   The communications between front end and back end have to be set up properly. The mechanisms, formats, and protocols regarding data transformation should be clearly defined.

6. Synchronization.
   The computation environment of the two sides needs to be synchronized. That is, when the front end user change the data, the back end executables should see the data change before the required computation, and visa versa.

We will address these issues in the following sections for and convey our experience for porting a stand-alone software analysis tool (CASRE) to a completely web-based tool (Web-CASRE).

3 A Stand-Alone Tool: CASRE

CASRE (Computer-Aided Software Reliability Estimation) system is implemented as a software reliability modeling tool with enhanced graphical user interface [3, 7]. CASRE was originally designed and implemented in a DOS Windows environment. It was subsequently ported into unix environment, using Tcl/Tk to re-implement its user interface. The command interface of CASRE is menu driven. Users are guided through the selecting of a set of failure data and executing a model by selectively enabling pull-down menu options. Modeling results are also presented in a graphical manner. After one or more models have been executed, the modeling results and comparisons are drawn in a graphics display window. Users can manipulate this window's controls to display the results in a variety of ways. Users may also display the results in a tabular fashion if they wish. CASRE executables for

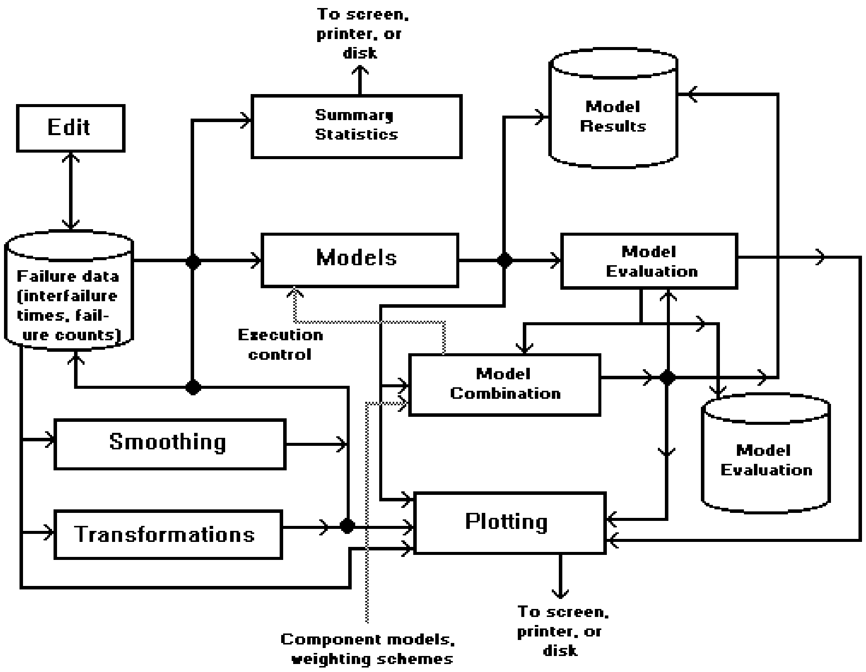

Figure 5: CASRE System Block Diagram

reliability models are implemented in Fortran and C modules. Figure 5 shows the high-level architecture for CASRE.

From Figure 5 we can see that the user of CASRE can perform edit function on the input data to and from a data base containing the failure data. This data set can be prepared by using any existing editors associated with the computer, or by invoking them within CASRE. CASRE can then perform filter functions on the data set. The filter functions, a pre-analysis utility, include transformations and smoothing. Summary of data statistics and plotting of data and modeling results are activated for both tabular format and graphical display. The user then can prepare, select, and apply software reliability models for the failure data. This is indicated in the box labeled "Models". The modeling results can be saved in the Model Results data base, and passed to the Model Evaluation function for model comparison. The results from Model Evaluation are saved in the Model Evaluation data base. The box labeled "Model Combination" is an optional facility allowing the user to define new models using linear combination of existing models, subject to their performance. At any stage when the data is available (either input or output), the plotting function can display the data graphically for clear examination.

In summary, CASRE comprises six major functional areas in its main window: File operations ("File" menu), Editing operations ("Edit" menu), Transformation and smoothing ("Filters" menu), Model selection and application ("Model" menu), Redraw display windows ("Redraw" menu), and Main window help system ("Help" menu). A graphics display window is invoked when the "Open" button under main window's "File" menu operation is selected. The graphics display window plots the failure data displayed in the work space and the results of applying models to that data. A separate menu bar is associated with this window, allowing users to control the contents and appearance of the display or send the contents of the window to an output device (disk file or printer). The menu items associated with this bar are: Ploting control ("Plot" menu), Modeling results selection ("Results" menu), Data and model results selection ("Display" menu), Display graphics control ("Settings" menu), and Graphics display window help system ("Help" menu).

4 Implementing Web-CASRE for Web Applications

With the existence of Tcl plugin [8], we use a special case of Figure 4 to implement Web-CASRE. This simplified architecture for web-based tool is seen in Figure 6. In the Unix version of CASRE we have implemented its user interface in Tcl/Tk language. Consequently, we use Tcl plugin version 2.0 in web browsers (Netscape and Microsoft Explorer) to download CASRE's front end GUI in the client side. In the server side we wrap Fortran executables, which implement the reliability model solutions, with C executables, and provide a dummy Tcl interpreter to interact with the C executables to preserve a global environment. This environment is defined by sharing the global states between the front end and the back end using sockets.

In the process of implementing Web-CASRE, we address the issues involved in web-tools, described in Section 2, as follows:

1. Storage of client data.
   Since the input data for Web-CASRE belongs to the user at the client side, we let user store all the data in his or her local disk. In the current implementation, propriety is deemed more important than performance. In fact, the performance of Web-CASRE in most cases do not suffer significantly with its applications.

2. Security concern.
   We have to assume that the server is trustable for the Web-CASRE application. This means the downloaded GUI code should be allowed to do most operations on the client side. To do this, we modify the Tcl plugin configuration file to allow Web-CASRE users to configure the use of policies, so that the trusted policy in the GUI code loaded from the Web-CASRE servers are allowed to execute in the client side.

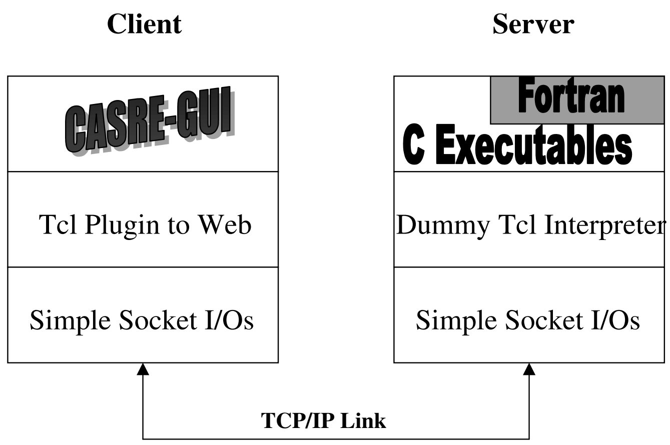

Figure 6: Web-CASRE Architecture

3. Interface simplification.
   We simplify the interface between front end and back end so that only one major computation, namely the model execution and comparison, is required for the back end executables. The front end GUI operations pack the input data and the globally required data structure in advance, submit them to the back end for model execution and comparison. The back end executables perform all the required computation, and send the complete results back to the front end. The GUI then can display the result data in various forms, using Tcl/Tk code loaded in the client side.

4. Front end computation.
   Simple Web-CASRE computations, including Edit, Smoothing, Transformations, Summary Statistics, Model Combination, and Plotting functions shown in Figure 5, are implemented for execution in the client side. Only the two major and most time-demanding computations, Models and Model Evaluation, are required for the server side.

5. Communications.
   The front end and the back end will set up a socket link for data transformation. A remote background process for Web-CASRE executables will run in the server, ready to accept the client who will open an appropriate socket channel. The front end GUI, knowing which server the executables run and what socket channel to open, can transfer data to the Web-CASRE back end for the required computation.

6. Synchronization.
   In order to synchronize the computation environment of the two sides, Web-CASRE contains procedures that are used to pack the global state into a Tcl script. This Tcl script prepares the execution environment with the required data and send the data as a character stream to the remote back end. The dummy Tcl interpreter in the back end unpacks the input data for the C and Fortran executables to perform command evaluations, and packs the computation results and the global state before transferring them back to the caller in the client side. Upon receiving the resulting global state, the front end will resume its operation on the data for various display utilities.

The resulting Web-CASRE can be executed by installing Tcl Plugin in the client side and by running a daemon process (for Web-CASRE back-end executables). The configuration for the Tcl Plugin should be modified to allow a trusted policy. To invoke Web-CASRE, users simply invoke a web page which includes:

```
<EMBED src="wcasre.tcl" width=420 height=270>
```

This command allows downloading the embedded `wcasre.tcl` file (for Web-CASRE front-end GUI) in the client's local disk, thus starting the GUI of this web-based tool.

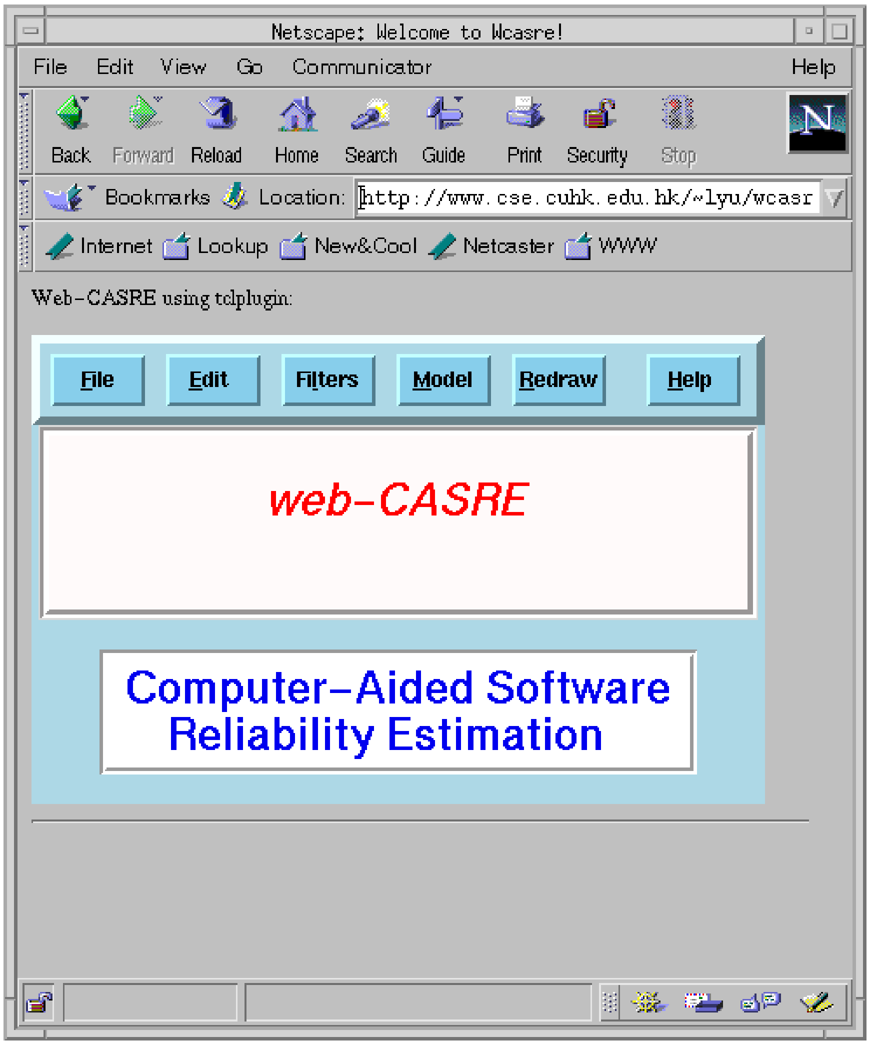

Figure 7: Web-CASRE Main Window Appearance

5 Web-CASRE On-Screen Appearances

When Web-CASRE is invoked, its main window and the associated manual bar are brought up as shown in Figure 7, using Netscape browser. From this figure we can see that the main CASRE operations, "File", "Edit", "Filters", "Model", "Redraw", and "Help" are available for the user to choose for operations. These manual buttons can also activated by their corresponding shortcut keys.

The user can first use File operation to open a data file. The data file is usually prepared by the user in advance and stored in the user's local disk. Figure 8 shows the action involved in opening a data file, where a pop-up window allows the user to navigate the directory structure for file selection, and the main window displays the input data in tabular format.

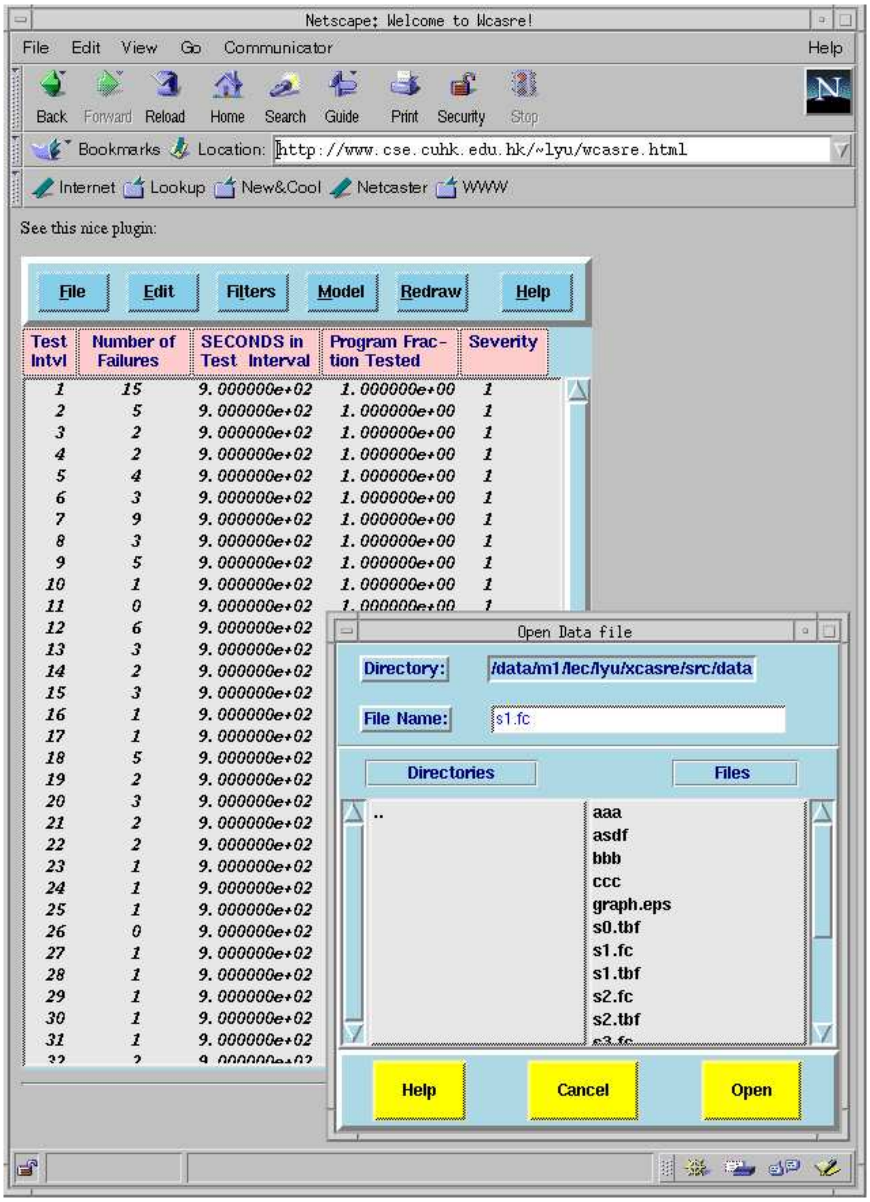

Figure 8: File Operation ins Web-CASRE

The graphical format of the input data will appear as soon as the input data set is selected. This is described in Figure 9. From Figure 9 we can see that a graphics display window (on the right-hand of the figure) plots the "failures per interval" input data. The manual buttons associated with this window, "Plot", "Results", "Display", "Settings", and "Help", are available to the user for display and printing control of this window. In this figure we also see that the user uses the "Model" operation in the main Web-CASRE window to select reliability models for execution. This is shown by the pop-up window below the graphics display window, where three models, Brooks/Motley Binary Model, Schneidewind Model, and Yamada S-Shaped Model are selected.

Figure 10 depicts the model execution results after applying the selected models to the input data. Instead of displaying the "failures per interval" data (shown in Figure 9) for display, the user has selected, using "Display" control, the "accumulated failures" data. The raw data and the results from the three selected models are plotted together for comparisons.

6 Conclusions

Web-embedded applications offer advantages over traditional applications in the following situations:

- Installing and maintaining the software is difficult or time-consuming.
- Porting the software to the user's computing platform is difficult, time-consuming, or impossible, if the software requires a computing platform that the user does not have, e.g., a massively parallel machine, a special-purpose operating system, or a proprietary data base.
- The software needs frequent updates. This is the case, for instance, with pre-release versions of software.
- The software requires a large amount of hardware resources but the user will need the software only rarely.
- The tool needs to be distributed quickly and conveniently a trial basis with restricted functionality.

In this paper we discussed the architectural, porting, security, and communications issues involved in creating a web-executable software tool. We examined the client-server architectures for the execution of web-based tools and laid out a procedure to convert a traditional stand-alone tool to a Web application. We illustrated how the Web-CASRE tool was created by using existing plugin facility associated with Tcl/Tk and web browsers. As the development and porting of such tools become more abundant and popular, the Web will eventually emerge from an information repository to a knowledge tool set, allowing net-centric computing to be realized in a cost-effective, systematic, and cooperative manner.

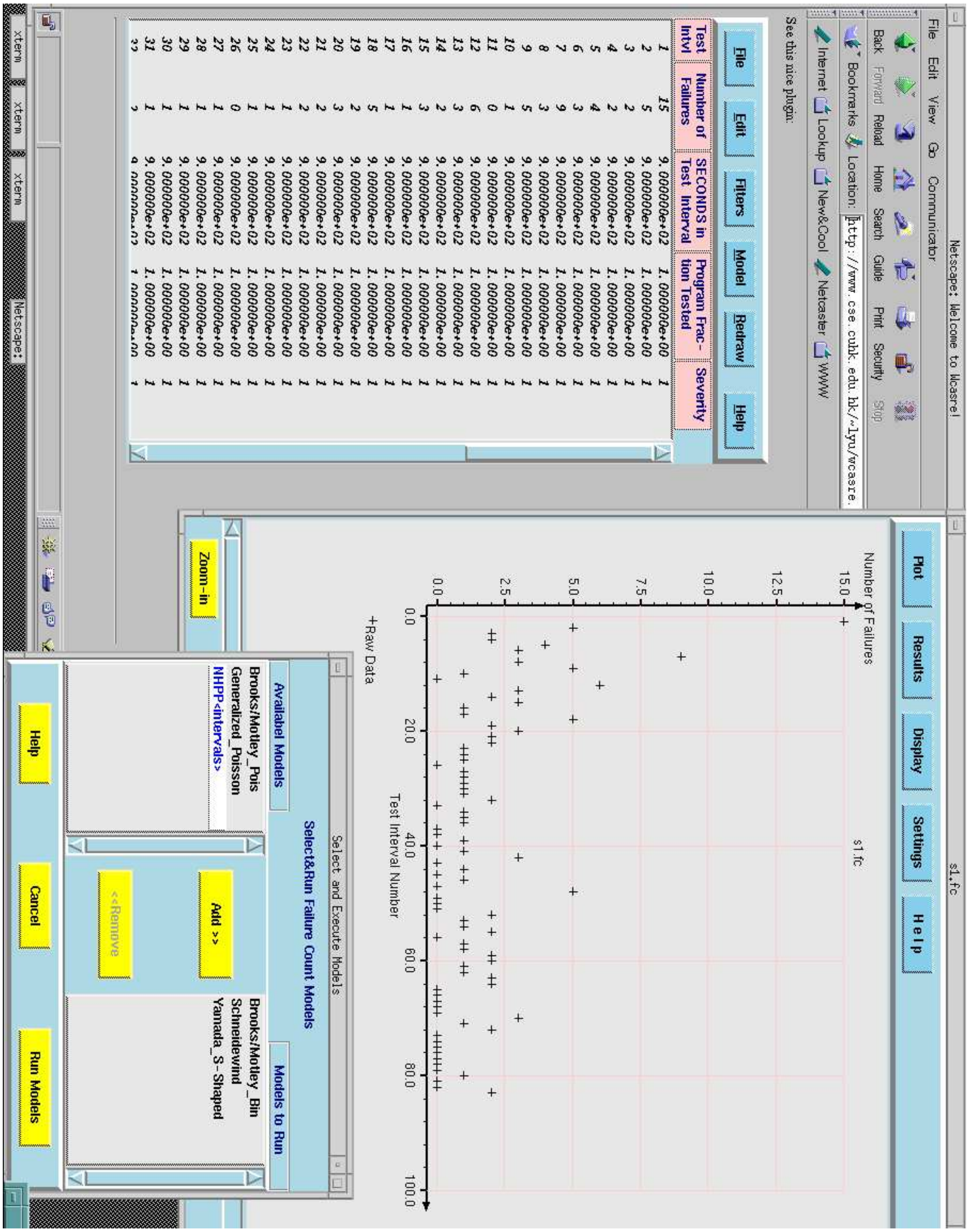

Figure 9: Data Display and Model Operation in Web-CASRE

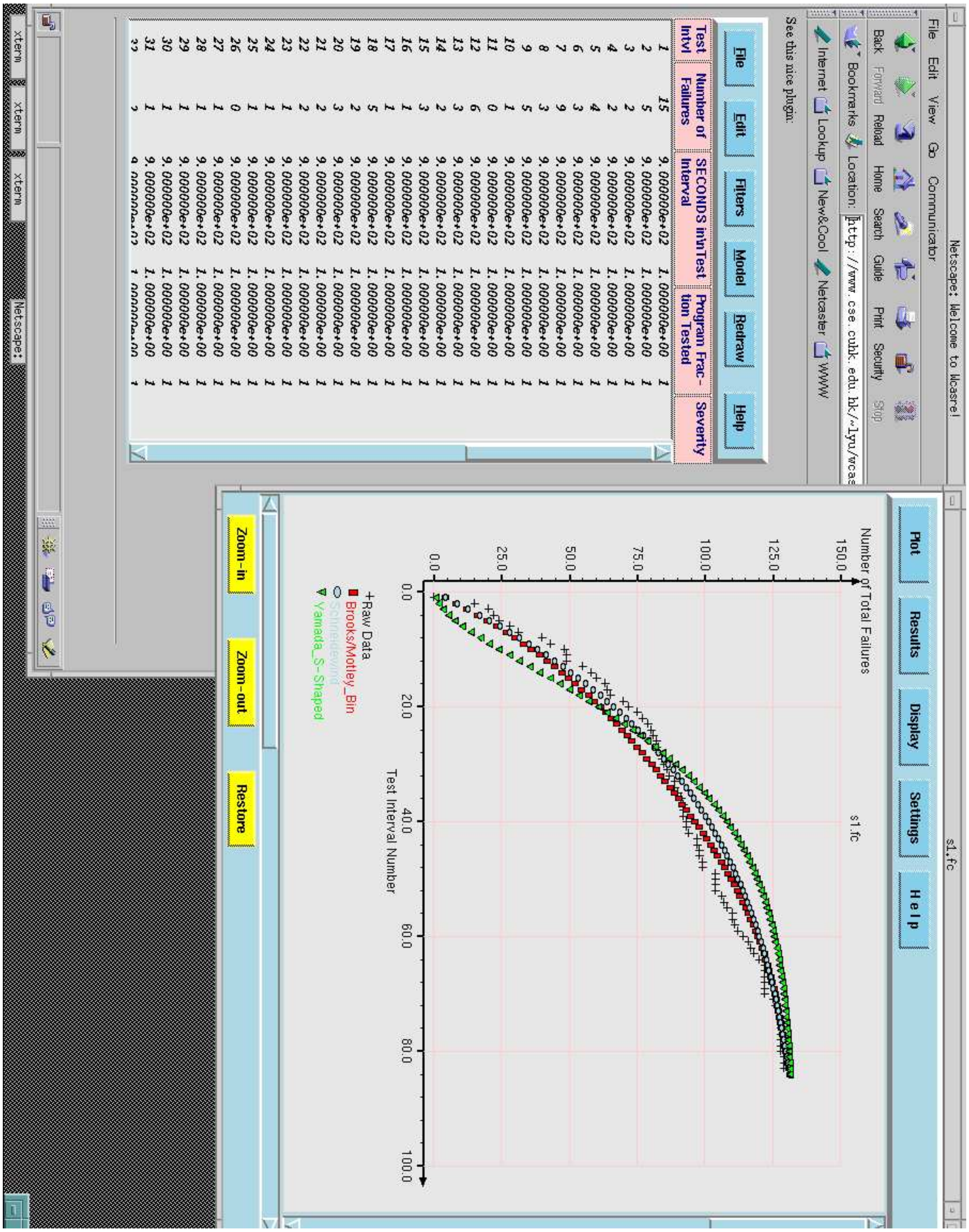

Figure 10: Model Results and Display Operation in Web-CASRE

References

[1] J. Domingue, P. Mulholland, Fostering Debugging Communities on the Web, *Communications of the ACM*, Vol. 40, No. 4, April 1997, pp. 65-71.

[2] A. Ford and T. Dixon, *Spinning the Web*, International Thomson Computer Press, 1996.

[3] M.R. Lyu and A. Nikora, "Using Software Reliability Models More Effectively," *IEEE Software*, pp. 43-52, July 1992.

[4] J.K. Ousterhout, *Tcl and the Tk Toolkit*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1994.

[5] A. Riva, M. Ramoni, "LispWeb: A Specialised HTTP Server for Distributed AI Applications," *Computer Networks and ISDN Systems*, Vol. 28, May 1996, pp. 935-961.

[6] R.W. Scheifler and J. Gettys, *heX Window System*, 3rd edition, Digital Press, 1992.

[7] G.E. Stark, "Software Reliability Tools," Appendix A of *Handbook of Software Reliability Engineering*, M.R. Lyu (ed.), McGraw-Hill and IEEE Computer Society Press, 1996.

[8] See `http://sunscript.sun.com/plugin/`.

[9] J. Trevor, R. Bentley, G. Wildgruber, "Exorcising Daemons: a Modular and Lightweight Approach to Deploying Applications on the Web," *Computer Networks and ISDN Systems*, Vol. 28, May 1996, pp 1053-1062.