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 problem solver. The design decisions regarding communications protocol, security policy, data storage, and consistent computation environment are also discussed in this paper.

1. Introduction

Since its first official proposal in 1989, the World Wide Web (or simply the Web) has drawn tremendous attention among not only academic professionals, but also computer illiterates [3]. 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 di-

*Supported by a Direct Grant provided from the Chinese University of Hong Kong.

rectly 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 problem solver 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 [2] servers supporting Web-embedded applications have appeared in the literature [1, 9, 13]. 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-Based Tools

A computation-intensive software tool is typically implemented in two major parts: its computation routines (the "problem solvers") and its graphical 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 problem solvers 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 on the local system, 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 can 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.

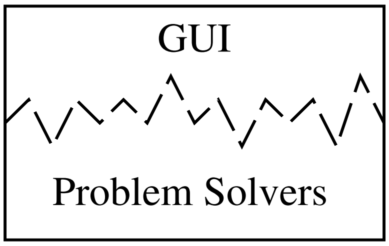

Figure 1. Typical Stand-Alone Software Tool Architecture

Figure 1 shows a typical stand-alone software tool. The GUI is what the user of the tool will see and operate, while the problem solvers, 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 problem solvers of a tool are usually implemented in different types of languages: The GUI uses graphics-enhanced protocols, languages and libraries like X Window [10], Windows Development Toolkit, Motif, or Tcl/Tk [6], while problem solvers are coded in efficient programming languages like C, C++, or Fortran. GUI and problem solvers are usually bundled together due to their high interactions. They share the same global environment and can transfer data and control flow between them conveniently. This often leads to a less structured interface between the GUI and the problem solver during the software implementation and maintenance phase, indicated by the dashed line in Figure 1.

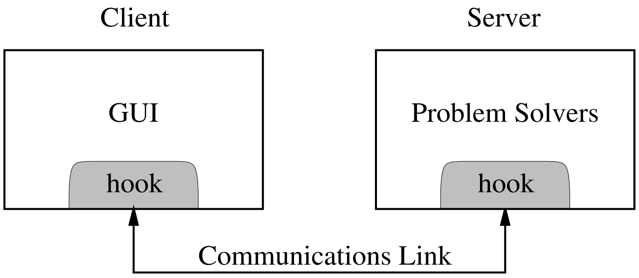

Figure 2. Client-Server Software Tool Architecture

In order to bring a stand-alone tool to the Web, the GUI and the problem solvers are separated, as seen in Figure 2.

After the separation, the GUI is running on a local site (the client site), and the problem solvers 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 problem solvers is shrunk into the hooks which link both sides.

Since the GUI and problem solvers in the original software are mutually involved, the hook in each side turns out to be a complicated layer. The originally bundled GUI-solvers pair share the same computation environment. This environment is no longer available, and these two parts will have to completely rely on their hooks to exchange messages. Creating these hooks in existing software can be very time consuming because of the tendency to violate the boundary between the problem solvers and the GUI in stand-alone implementations.

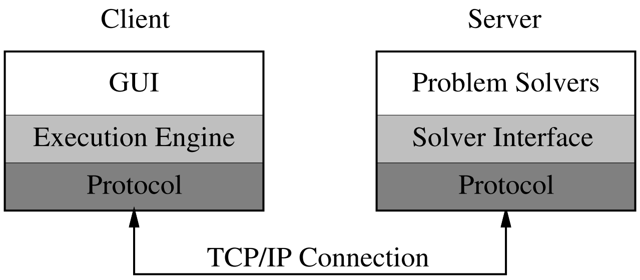

Figure 3. Web-Based Software Tool Architecture

Figure 3 shows the Web-based architecture, where the GUI on the client site will run on top of a standard execution engine which offers code portability. A typical example is the Java virtual machine [4], which is integrated into many popular Web browsers. Communication between the client and the server is usually implemented by a special purpose communication protocol. One of the reasons for choosing a non-standard communication protocol is the stateless nature of HTTP, which makes HTTP more difficult to use than a special purpose statefull protocol. The communication protocol is usually running on top of a TCP connection over IP. The problem solvers have to be wrapped properly by well-defined interfaces to control the execution and data access from clients.

Implementing the Web-based software architecture shown in Figure 3 requires to address the following design issues:

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

2. Security concern.
   When the client downloads user interface code from the server, there may exist security holes if the server is not trustworthy. When all the commands of the GUI code are allowed to execute, malicious action can easily invade the system. However, if no potential risky commands are allowed, the GUI code becomes useless for sophisticated functions.

3. Interaction simplification.
   In order to limit the expensive communications between the GUI and problem solvers, 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 problem solvers 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 on the client 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 must be clearly defined.

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

We address these issues in the following section and convey our experience for converting a stand-alone software reliability analysis tool (CASRE) into a 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 [5, 11]. 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

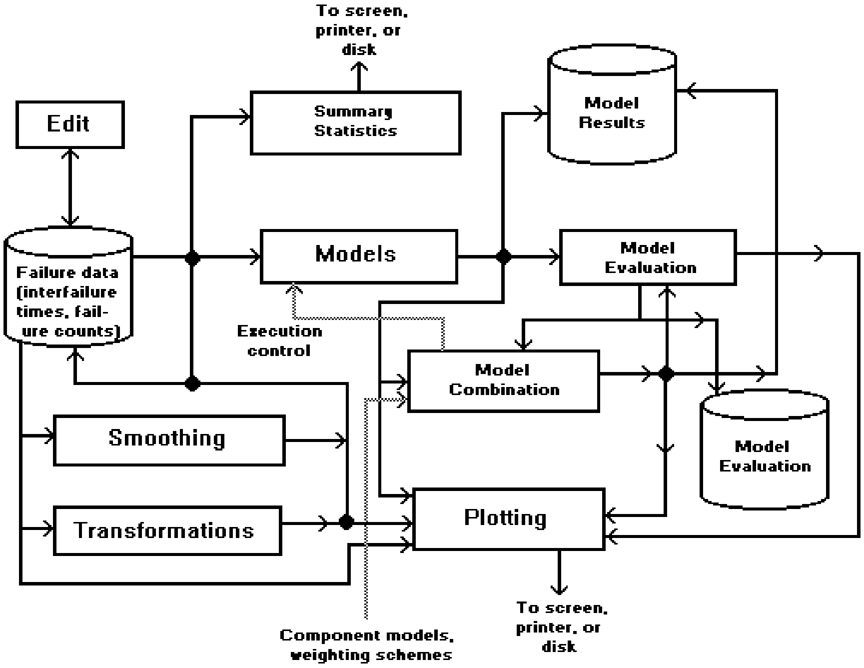

Figure 4. CASRE System Block Diagram

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 reliability models are implemented in Fortran and C modules. Figure 4 shows the high-level architecture for CASRE.

From Figure 4 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 a Web-Based Tool: Web-CASRE

The Unix version of CASRE implements the GUI in the Tcl/Tk language. With the existence of the Tcl/Tk plugin [12], we decided to use a special case of Figure 3 to implement Web-CASRE, shown in Figure 5. The Tcl/Tk plugin version 2.0 is available for the most popular Web browsers, namely Netscape and Microsoft Explorer.

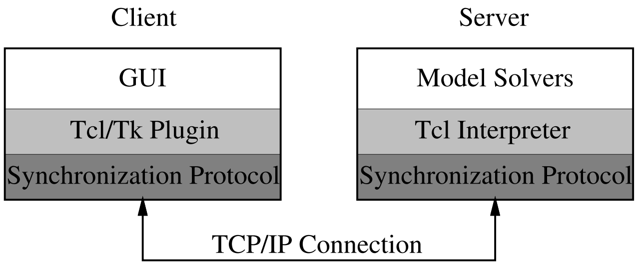

Figure 5. Web-CASRE Architecture

An analysis of the Unix implementation showed that the boundary between the problems solvers and the GUI was not well structured. In order to avoid a major rewrite to separate both parts cleanly, we decided to try a solution which works without knowing the details of the interactions between the GUI and the problem solvers. The basic idea was to synchronize the interpreter on the server side with the interpreter implementing the GUI. This approach ensures that the problem solvers function as if the GUI is locally available and that the GUI functions as if the problem solvers are locally available. The implementation of this solution turned out to be quite small, due to the reflective nature of Tcl. However, fine-grained synchronization could not be achieved because Tcl does not allow to trace variable creations. This forced us to implement a coarse-grained synchronization, which causes some communication overhead.

In the process of implementing Web-CASRE, we addressed the design issues described in Section 2 as follows:

1. Data Storage.
   Since the input data for Web-CASRE belongs to the user at the client side, we let user store all the data on his or her local disk. In the current implementation, propriety is deemed more important than performance. In fact, the performance of Web-CASRE does not suffer significantly with increasing data sets if there is a reasonably fast network connection between the server and the browser.

2. Security concern.
   We have to assume that the server is trustworthy. 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 enable the "trusted" security policy for the GUI code loaded from the Web-CASRE server. This allows to execute security sensitive commands in the Tcl/Tk plugin. The Tcl/Tk plugin is shipped with a very restrictive default security policy [7] which even inhibits the Tcl/Tk code to create menus or toplevel windows. The default security policy is therefore almost useless for any serious application.

3. Interaction simplification.
   We simplify the interactions between the front end and the back end so that only one major computation, namely the model execution and comparison, is performed by the back end problem solvers. The front end GUI packs the input data and the globally required data structure and submits them to the back end for model execution and comparison. The back end problem solvers 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 into the client.

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

5. Communications.
   The front end establishes a TCP connection to the back end to synchronize the Tcl interpreters. A remote background process realizing the Web-CASRE problem solvers runs on the server, ready to accept clients who will open a TCP connection. The front end GUI, knowing which server the problem solvers runs on, can transfer data to the Web-CASRE back end for the required computation. The data is send as a character stream, which is basically a Tcl script which transfers the state of the client's Tcl interpreter to the server's Tcl interpreter and back.

6. Synchronization.
   In order to synchronize the computation environment of the two sides, the Web-CASRE front end contains procedures that are used to pack the state of the Tcl interpreter into a Tcl script. This script prepares the execution environment on the back end. The Tcl interpreter in the back end unpacks the input data for the C and Fortran problem solvers to perform command evaluations, and packs the computed results and the state of the back end interpreter before transferring them back to the front end. After reconstruction of the result state in the

front end interpreter, the front end will resume its operation on the data for various display utilities.

Web-CASRE is invoked by down loading an HTML [8] page which an embedded HTML tag that refers to the Tcl/Tk code implementing the GUI. The Web browser will automatically load the Tcl/Tk plugin into the Web browser and download the GUI when the HTML page is loaded. No special user interactions are needed to start to tool once the Tcl/Tk plugin has been installed on the client side.

5. 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.

References

[1] J. Domingue, P. Mulholland, Fostering Debugging Communities on the Web, *Communications of the ACM*, Vol. 40, No. 4, pp. 65-72, April 1997.

[2] R. Fielding, J. Gettys, J. Mogul, H. Frystyk and T. Berners-Lee, *Hypertext Transfer Protocol – HTTP/1.1*, RFC 2068, UC Irvine, DEC, MIT/LCS, January 1997.

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

[4] T. Lindholm and F. Yellin, *The Java Virtual Machine Specification*, Addison Wesley, 1997.

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

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

[7] J.K. Ousterhout, J.Y. Levy and B.B. Welch, *The Safe-Tcl Security Model*, Technical Report, Sun Microsystems Laboratories, November 1996.

[8] D. Raggett, A. LeHors and I. Jacobs, *HTML 4.0 Specification*, W3C Recommendation, December 1997.

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

[10] R.W. Scheifler and J. Gettys, *The X Window System*, 3rd edition, Digital Press, 1992.

[11] 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.

[12] `http://sunscript.sun.com/plugin/`.

[13] 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, pp. 1053-1062, May 1996.